DUAL PROCESSOR SERIAL NETWORK

BY

TIINA ZEIN

B.S., University of San Francisco, 1982

ACKNOWLEDGEMENTS

Many students became involved in this project and I would like to express my gratitude to them. For the design of the printed circuit boards, I would like to thank John Coble and Tim Harvey. For his support and motivation, a special thanks to my thesis advisor, Professor Ricardo Uribe. Finally, I would like to thank Sami Zein who has applied the network to a robot arm. He has given many valuable ideas for my work.

TABLE OF CONTENTS

APPENDICES

CHAPTER 1

INTRODUCTION

The Advanced Digital Systems Laboratory (ADSL) Dual Processor Serial Network has a long history. Several students have contributed to the development of a powerful network. However, when the present work started, the network did not really function as a network. The node-to-terminal connections were manual. The node-to-node connections didn't exist.

The goal of this thesis has been to develop the existing network so that it becomes a flexible and user-friendly developmental tool. All the connections from one node to another or to the terminal are made in the software. A flexible master-slave hierarchy has been created. The user has the freedom to interpret this hierarchy and the only limitation is that the highest master has priority to access the terminal.

Every node board has two 8751 microcomputers, the Central Processing Unit (CPU) and the Input/Output (IOP) processors. The 8751 has a 128 byte RAM and a 4K byte on chip EPROM where the monitor programs are loaded. Both processors have different monitor programs. Most of the monitor program is in the IOP processor. The CPU processor has a program to control the 8155

port expander. The remaining on chip EPROM of the CPU processor can be used for user programs.

The network has the capacity for 20 node boards, but only four prototype boards exist now. Printed circuit boards, which will be available soon, will ease significantly the growth of the network.

CHAPTER 2

HARDWARE

The hardware of the network includes two cardcages, two terminal interface boards, four dual processor boards (nodes) and the Hazeltine terminal. The backplane of the cardcages connects the node boards to each other and to the terminal through the terminal interface board.

2.1 The Cardcages

The network is contained in a frame of two cardcages built one above the other. The frame of the cardcages is shown in figure 1. Both cages are independent of each other and have their own interface boards to the terminal.

Each cage consists of 11 card slots. The right-most card slot is reserved for the terminal interface board in both cages. The 10 remaining card slots are for node boards. Due to the way the software is written, the node boards must be placed in the cardcage starting from the left and leaving no empty slots between boards.

The power supply is located under the cardcages. The supply

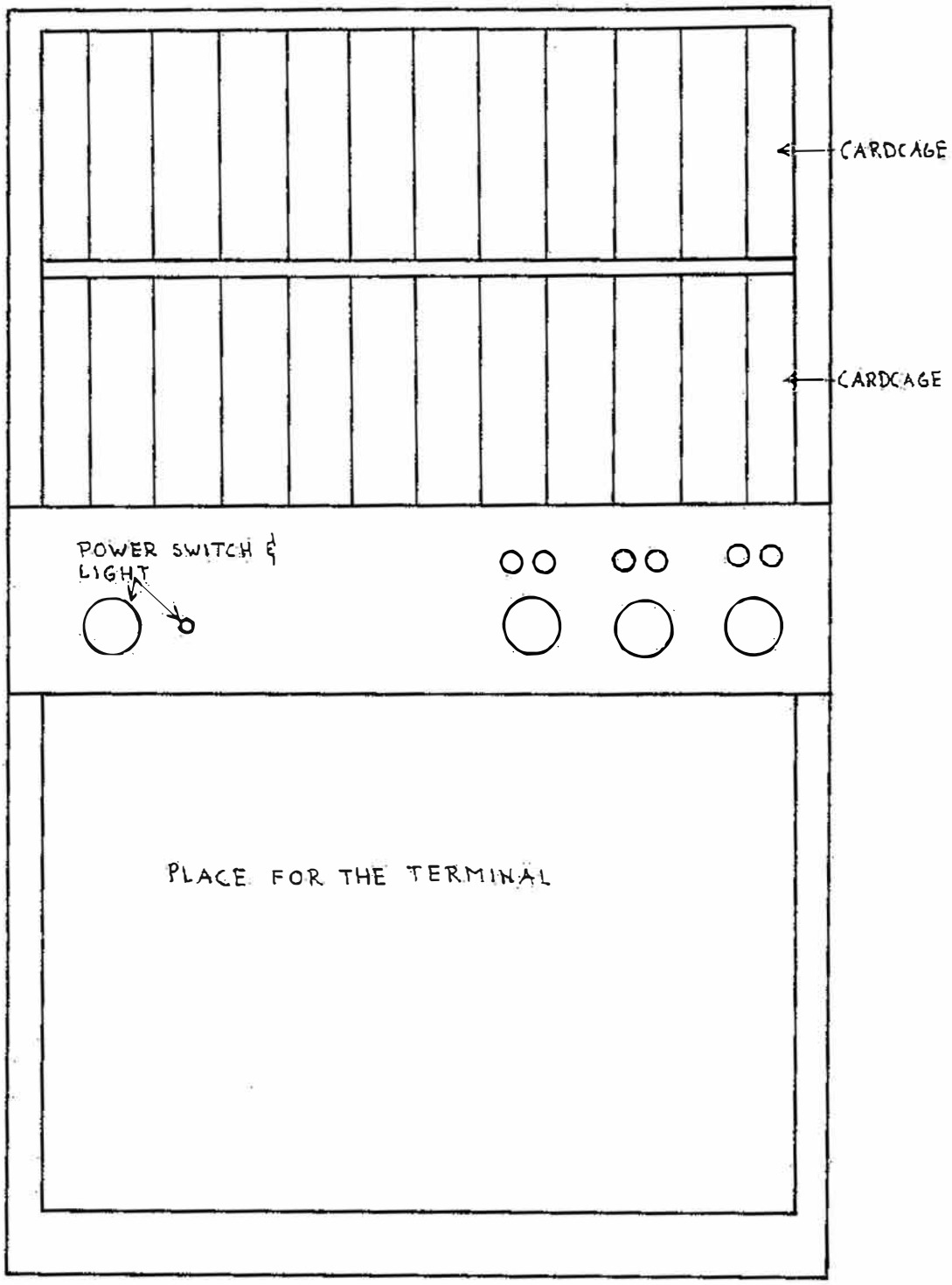


Figure 1. The frame of the cardcages

offers +5 volts, +12 volts and -12 volts. A voltage of +5 is needed for all boards. The supply also incorporates a sensing circuit to maintain +5 volts beyond the terminals of the power supply. These sensing wires are connected to the same pins at the backplane connector as are +5 volts and ground. The +12V and -12V are used in the interface board to generate standard levels for the Hazeltine terminal.

The network has two power switches, one on the front panel and one on the back panel, which allows power to be turned on and off easily on both sides. In order to turn on the power, both switches must be turned on, which gives some extra moments for the user to double check everything before the power is on. Ligths are also provided for each voltage on the front panel.

The node-to-node connections and node-to-terminal connections are on the backplane. Each connection has two lines to allow a simultaneous transmission and reception of data. Figures 2a and 2b show these connections. Connectors in all slots are drawn with pins marked from 1 to 22 and from A to Z. The connections to pins 1-22 are shown on the left side of the connector and to pins A-Z on the right side of the connector. Each connection has first the slot number and then the pin number. For example, pin 7 of the connector in slot #1 is connected to slot #2, pin F.

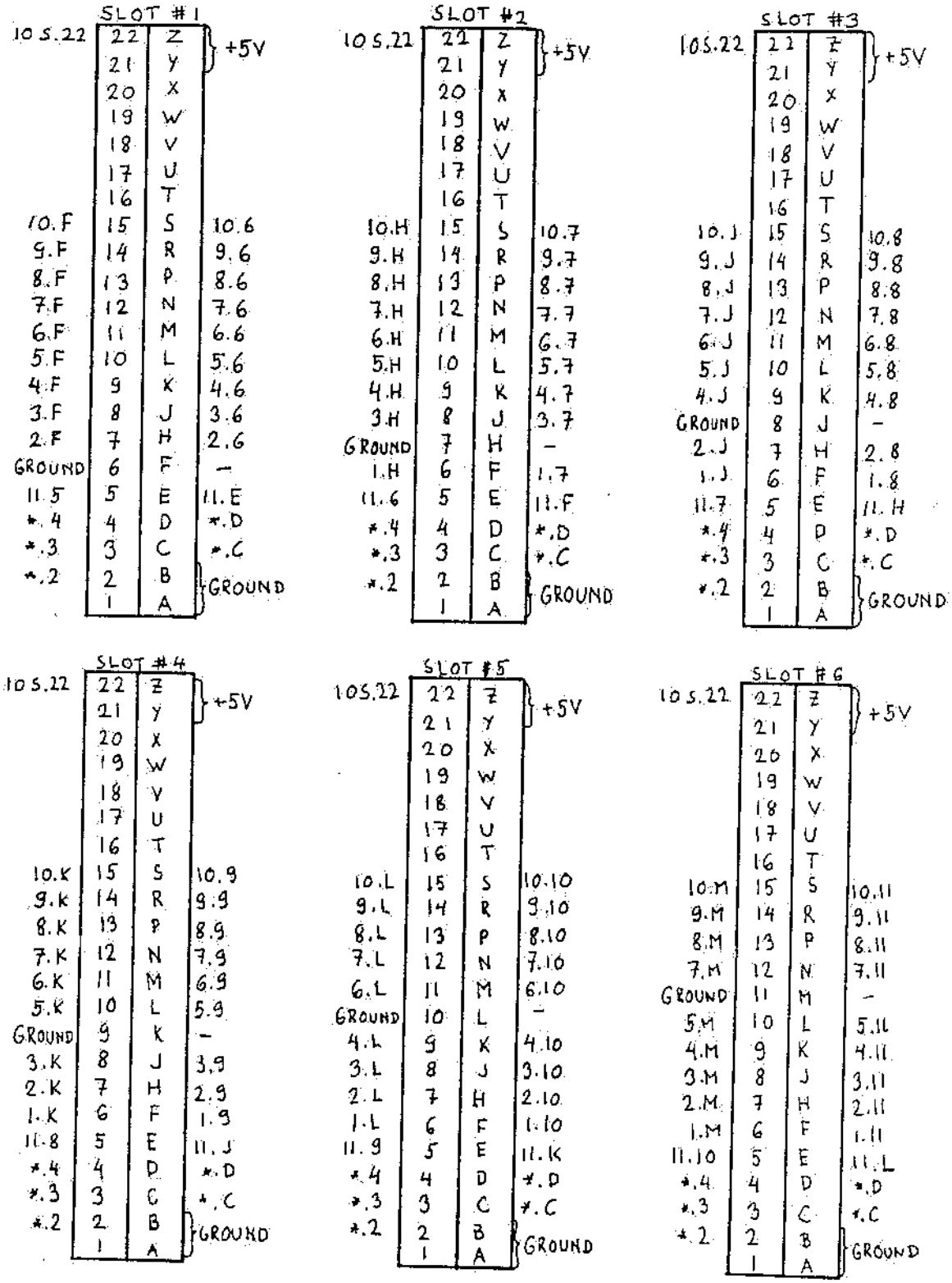


Figure 2a. Slots #1-#6

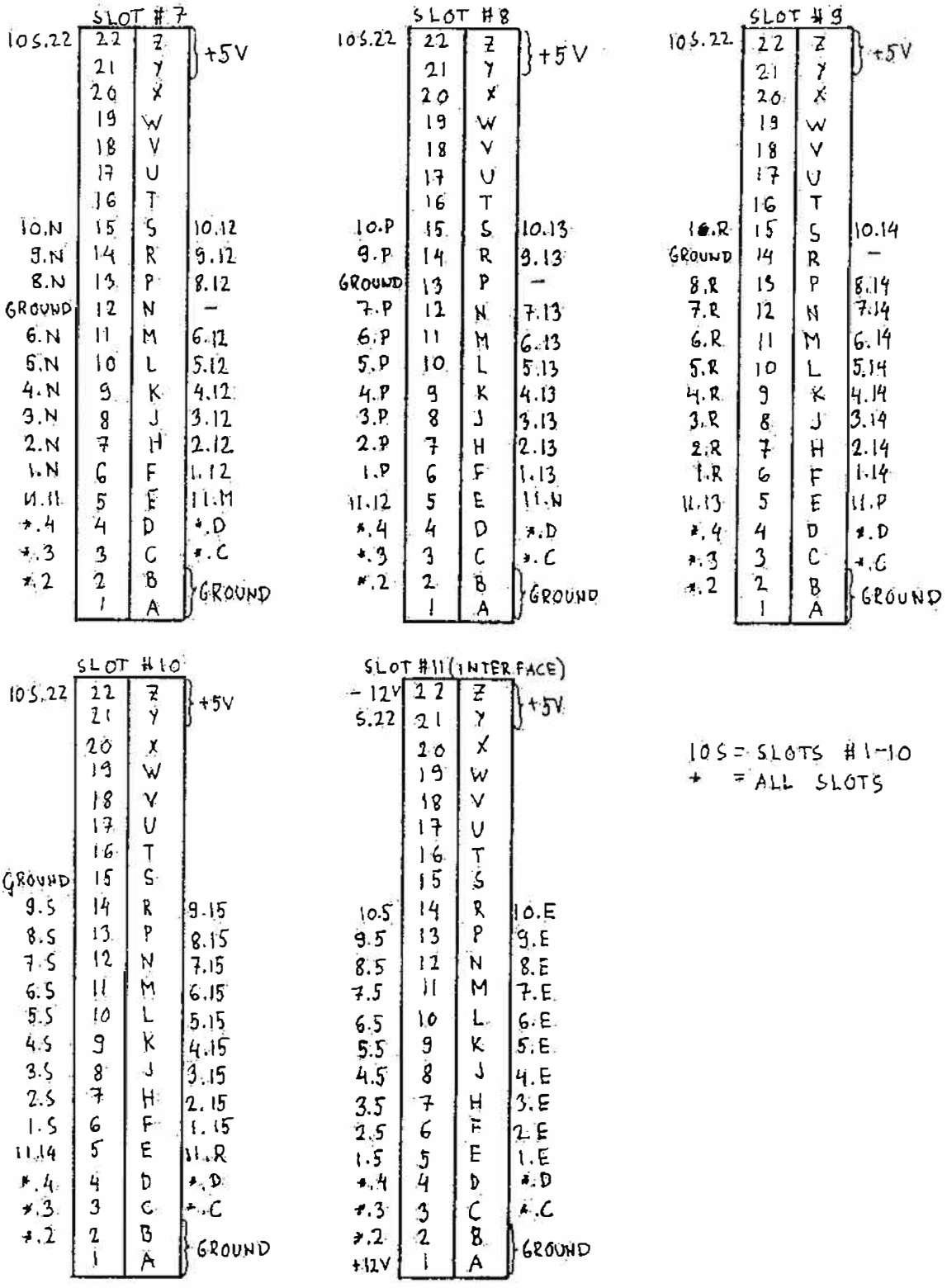


Figure 2b. Slots #7-#11

2.2 The Terminal Interface Board

The terminal interface board (see Appendix A) is always placed in the right-most card slot in the cardcage. It receives +5 volts at pins Y and Z in the backplane and +12 volts comes to pin #1 and -12 volts to pin #22. The ground line is at pins A and B.

Pin 5 of every card edge connector is connected to the interface board. These lines function as transmitting lines from the terminal to the node boards. In the same manner, pin E of every backplane connector is connected to the interface board in order to transmit data from a node board to the terminal.

The original design had a hex switch that selected the node to be connected to the terminal. The manual selection of the node-to-terminal connection was a limiting factor for the development of the network. It would have forced a simple hierarchy of one master. The hex switch was removed to make the network more flexible. The backplane pins 2,3,C and D of all node boards are connected to these same pins at the interface board. These are the P1.4-7 of the IOP processors. They are also connected to the multiplexer and demultiplexer of the interface board to select the desired line. Thus, the hex switch is omitted. The node-to-terminal connection is made in the software through the P1.4-7 lines.

The network reset is at pin 4 in the backplane and it is connected to all nodes. This line resets the IOP and the CPU processors and the 8155 of all boards simultaneously. Moreover, even if the terminal is not used, the interface board must be connected to prevent the reset pin from floating. The reset pin is normally zero.

The terminal connection from the interface board uses a standard RS232C serial connector. Pin 2 transmits information to the terminal and pin 3 receives from the terminal. Pins 1 and 7 are grounded. Another RS232C connector connects the terminal interface board to the Intel Personal Development System (iPDS).

2.3 The Dual Processor (Node) Board

The dual processor board (see Appendix A) consists of two 8751 microcomputers, node memories and logic that selects the desired memory location, multiplexer and demultiplexer for the serial connection, crystal circuit, reset circuit and two connectors to receive and transmit data to and from the outside of the network.

The IOP processor can select either the terminal or the node it wants to talk to. This is done by using Port 1 pins 0-3. In the terminal connection pins P1.4-7 must also be properly set to connect the terminal with the node. The access from other

nodes or from the terminal directly to the CPU processor of a node is a new feature in this network. The RXD pins and TXD pins of both processors are connected together on the node board to make the serial lines equally available for both processors. The serial control register bit REN (enable serial reception) is set and cleared in the software so that only one processor at a time uses the serial lines. The CPU does not have the capability to select directly the target of its communication. It must be selected by the IOP processor.

Ports P0 of both processors are connected together to form an internal bus. The lower bytes of addresses and data move through this bus. The CPU has the priority to use it. The IOP processor must interrupt the CPU and request the bus before it can use it.

Port P1 of the IOP is used to select to whom the node or the terminal is going to talk to. Pins P1.0-3 select the target for the node's own data transfer. The pins P1.4-7 of all IOPs, connected as a bus in the backplane, select in the interface board the node to which the terminal is going to talk to. Port P1 of the CPU has a different function. It can be connected to the outside world through a 20 pin Scotchflex connector. The printed circuit boards will include a bidirectional buffer between Port P1 and the connector.

Port P2 pins 0 to 6 select high byte addresses. The P2.7 pins of both processors are connected together to form the bus

grant or interrupt acknowledge line. The bus request line is between interrupt 0 of the IOP and interrupt 1 of the CPU.

The design of node memories has not been efficient. The processors could address 32K bytes, but only two 1K RAMs, one 2K EEPROM, and a 256 word x 8 bit 8155 port expander are available. The new printed circuit boards will provide 24K of node memories.

The port expander 8155 has two 8 bit ports A and B and one 6 bit port C. All ports can be used as inputs or outputs. Port C can control ports A and B. The memory address is latched with the falling edge of ALE from the processors. The port expander has a 14 bit timer and a command register. One can only write to this register but not read it.

The 8185 is a 1K static RAM. Its advantage is low power dissipation.

The 2817A is a 2K byte EEPROM. It has a fast read access time of 250 ns, but writing is slow. The write time is 5 ms and the cycle time 10 ms.

A 21 MHz crystal with a 74LS629 oscillator is used. The output of the oscillator is connected to the input of a 74LS393 counter to reduce the frequency to about 7 MHz. Two crystals are needed - one for each processor. Since an outside crystal is used, it must be connected to the crystal 2 pin of the processor and the crystal 1 pin must be grounded.

In addition to the network reset, each node has its own

reset circuit which resets both processors and the 8155. This reset is connected through the backplane pin 4 to the master reset on the terminal interface board.

The 74LS154 multiplexer uses P1.0-P1.3 to select one of the sixteen output lines and transmits the serial message to this line. The 74LS150 demultiplexer receives the serial message on the line selected by P1.0-P1.3 and transfers it through an inverter to the processor. Even though there are 16 lines available, only ten of them are used in this network.

2.4 Hazeltine Terminal

It is important that the switches in the front panel of the terminal are set correctly so that the transmission and the reception of data with the network can occur.

The baud rate of the terminal should be set to 9.6K. This is the same baud rate as the network uses.

From the four parity switches, 0 must be selected. The 0 parity means that the parity bit of each character transmitted is set to a zero. No parity check is done on received data.

The upper/lower case operation (U/L) is used and the characters are displayed white on the black screen (STD VIDEO).

CHAPTER 3

THE MONITOR PROGRAM FOR THE IOP

The monitor program for the IOP fills almost entirely the 4K program space of the processor. The flow chart of the IOP monitor program is shown in figure 3. The processor is initialized for the serial communication. Each node finds its own slot number and from that number several other numbers are calculated. These numbers are used for the writing of the master-slave table. The user can then choose any node to enter commands. The monitor program includes several small procedures that are also in the CPU program. These procedures involve mainly serial reception and transmission of data. They may be moved to a node memory if the user needs more space in the IOP. The procedure BUS REQUEST is very important for the IOP, since the CPU normally has the bus.

3.1 Initialization of the IOP Processor

The monitor program for the IOP processor starts with several initializations of the processor. The timer/counter mode control (TMOD), the serial port control register (SCON)

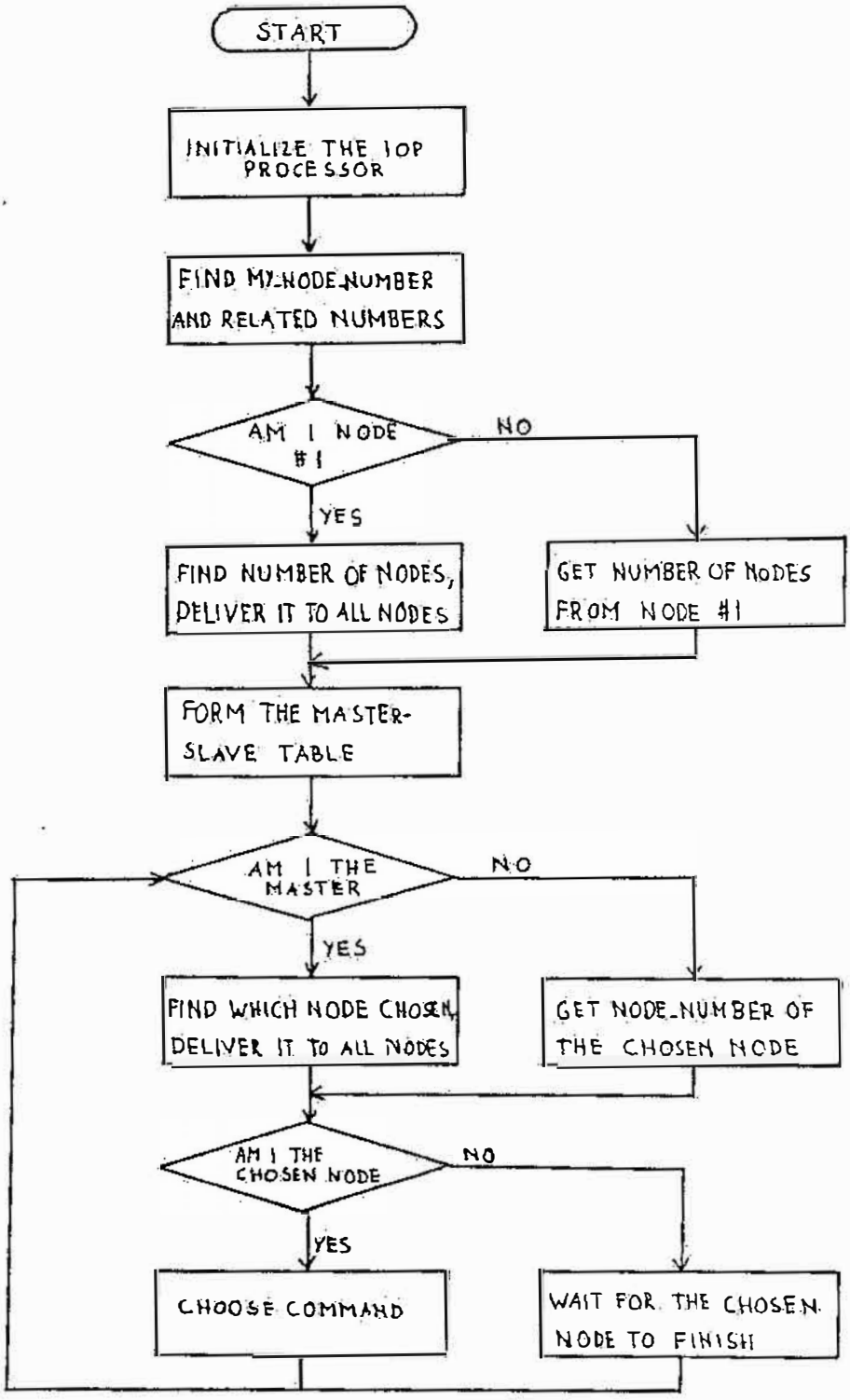


Figure 3. Flow chart of the IOP monitor program

and serial interrupt enable (ES) must be given some starting values (see reference 1).

The timer 1 is initialized to 8-bit auto re-load mode. The timer/counter mode control register TMOD has 8 bits (see figure 4). The four most significant bits define timer 1 and the four least significant bits control timer 0.

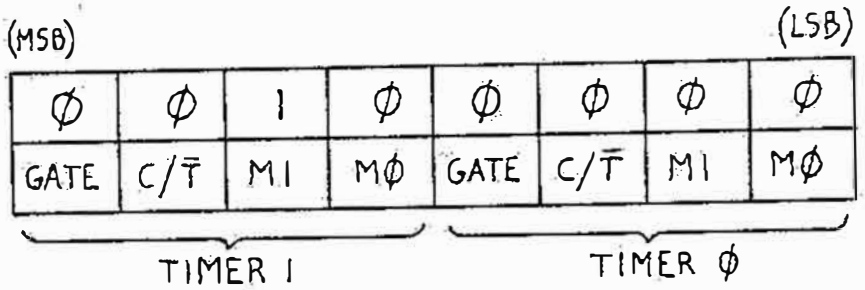


Figure 4. TMOD

When the gate controls are cleared, the timers are enabled whenever the respective TRx control bits are set. The TR1 of timer 1 is set.

The serial port control register SCON is initialized to 8-bit mode 1 (see figure 5).

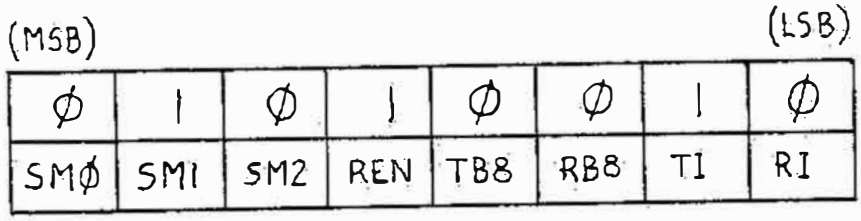


Figure 5. SCON

SM0 and SM1 define the mode used. In this case mode 1 is chosen. Ten bits are transmitted or received: a start bit (0), 8 data bits (LSB first) and a stop bit (1). In mode 1 the SM2 bit can be used to check the validity of the received stop bit. SM2=0 omits this feature.

REN enables serial reception. In the IOP, REN is originally 1 and in the CPU it is 0. When the CPU is interrupted to receive serial information, the REN bit of the CPU is set and it is cleared in the IOP.

In mode 1 the TB8 is not used. RB8 is the received stop bit.

TI and RI are the "transmit interrupt flag" and the "receive interrupt flag," respectively. They are set by the hardware but they must be cleared by the software.

The serial interrupt (ES) is part of the Interrupt Enable register (IE). ES is cleared to disable the serial port interrupt.

3.2 Numbers Related to the Position of the Node

All IOP processors have the same monitor program. However, every IOP needs variables that indicate the position of the node in the cardcage. The values of these variables are found in the program. Every node has its own receive line connected to ground. At power on, all other receive lines are high. Each node goes through the receive lines until it finds a zero. The

four low bits of Port 1 indicate the position of the node in the cardcage. In the software this position is called MY-NODE-NUMBER.

Before a master is determined, node #1 functions as the master communicating with the terminal and collecting and transferring information to other nodes. When a node communicates with the terminal, two conditions must be met: the node must be connected to the terminal and the terminal must be connected to the node. The first condition is met by inserting zeros to the four least significant bits of P1. The four most significant bits determine the connection of the terminal to the node. The connection to node #1 is made by defining P1.4-7=NOT(0000), to node #2 P1.4-7=NOT(0001), to node #3 P1.4-7=NOT(0010), etc. One has to know the hardware of the node board and the interface board in order to understand the generation of these numbers in the software. Each node finds its own terminal number for the node-to-terminal connection. For example, for node #9 MY-TERMINAL-NUMBER is found as follows:

(MY-NODE-NUMBER) - 1 = 9 - 1 = 8 = 1000B; B=binary

NOT(MY-NODE-NUMBER - 1) = NOT(1000B)= 0111B

MY-TERMINAL-NUMBER = SHL(0111,4)= 01110000B; SHL=Shift Left

The network can logically have 15 nodes and the interface

board in a cardcage even though there are now only ten slots for the nodes available. The software is written for 15 nodes keeping in mind possible future expansions. These 15 nodes are arranged in a master-slave table using the four high bits of P1. This table arranges all nodes on the basis of how fast they are. The fastest node will be the highest master. Other nodes follow in the order in which they qualify. This table has only one specific function: the node on the first position in the table will be connected to the terminal. It is up to the user to decide how the rest of the table is interpreted. A possible way to use it is to assign each node as a slave to the nodes above it and as a master to the nodes below it. This table starts at the address 9E00H and it can be changed at any time using the command CHANGE TABLE. This command is explained later with other commands.

Since there are more nodes than P1 lines, for each element in the master-slave table the P1 lines must be used twice during the generation of the table. Figure 6 shows how the P1 lines are divided among the nodes.

First the lines are used by the nodes in groups of 4, 4, 4 and 3. For example, the nodes 1-4 use line P1.4. In the software this is called MY-VOTE1-NUMBER. MY-VOTE1-NUMBER for the node #9 is 6. This is found as follows:

(MY-NODE-NUMBER) - 1 = 9 - 1 = 8 = 1000B

SHR(MY-NODE-NUMBER - 1,2) = 0010B = 2; SHR=Shift Right

MY-VOTE1-NUMBER = 4 + SHR(MY-NODE-NUMBER - 1,2) = 4 + 2 = 6

Since the node numbers start from one instead of zero, each MY-NODE-NUMBER is first decremented by one. This makes it easier to find a pattern to form groups of four nodes.

node #	P1 line used at first trial	P1 line used at second trial
1	4	4
2	4	5
3	4	6
4	4	7
5	5	4
6	5	5
7	5	6
8	5	7
9	6	4
10	6	5
11	6	6
12	6	7
13	7	4
14	7	5
15	7	6

Figure 6. P1 lines used by nodes to form the master-slave table

At the second trial, all the nodes with a specific MY-VOTE1-NUMBER can try again. MY-VOTE2-NUMBER is found for the node #9 as follows:

(MY-NODE-NUMBER - 1) AND 0011B = 8 AND 0011B = 0000B

MY-VOTE2-NUMBER = 4 + (8 AND 0011B) = 4

3.3 Number of Nodes

Node #1, as a temporary master, is first connected to the terminal. The following text will appear on the screen:

ENTER ZERO FOR CONNECTION

A zero is expected before the program goes on. However, any character will be accepted as an answer.

Next, node #1 will ask:

HOW MANY NODES

The number of nodes in use must be entered. If there are four nodes in use and 02 (two characters are expected) is entered, only the nodes #1 and #2 will be in the master-slave table. Node #1 will send the number of nodes to the other nodes. On the other hand, if 06 (>four) is entered, the program will fail and reset will be necessary.

While node #1 talks with the terminal, all other nodes connect themselves to node #1 without affecting the four most significant bits of P1. The timing is very important in node-to-node connections. The node receiving data must be ready to receive before the data are sent. Since node #1 is talking with the slow terminal data stream, the other nodes have enough time to connect themselves to node #1 and wait for the incoming data.

3.4 Master-Slave Table

The numbers related to the position of the node are used for the master-slave table which is created next. This table will include as many nodes as the user has entered earlier. If the user has entered three nodes, nodes 1,2 and 3 will be in the master-slave table. The table is found at the addresses 9E00 to 9E00+(number of nodes)-1.

While the master-slave table is formed, the P1.4-7 lines are momentarily given another function than to connect a node to the terminal. These lines are all set high. As long as all these lines are set, a node can try to clear the P1 line specified with its own MY-VOTE1-NUMBER. From figure 6 one can see that node #9 has MY-VOTE1-NUMBER=6. This means that node #9 tries to turn off the P1.6 line.

Node #1 goes through the P1.4-7 lines starting from P1.4. When it finds the first zero, it stops and sends the number of that line to all participating nodes. Those nodes that recognize the received number as their own MY-VOTE1-NUMBER have another chance to try to be the highest master.

The second time is basically the same as the first time except no more than four nodes can participate and each of them uses a different P1 line. For example, if all nodes whose MY-VOTE1-NUMBER=6 are trying the second time, then node #9 uses P1.4 line, node #10 uses P1.5 line, node #11 uses P1.6 line and

node #12 uses P1.7 line.

Node #1 again goes through the P1.4-7 lines in order to find out which line is cleared first. From that data, node #1 calculates the node number of the winner. It sends the number of the winner to all nodes. Each node writes this number to the first place in the master-slave table.

This same procedure is repeated to fill the whole master-slave table. Those nodes that are already written to the table will clear their DATA-FLAG bits. This will prevent them from being written twice in the table.

When the master-slave table is ready and all nodes have a copy of it, the highest master takes over the terminal connection from node #1. The program in all nodes then moves to the procedure WHICH-NODE.

3.5 Which Node

The new master will now print to the terminal screen:

WHICH NODE:

This question must be answered with a two digit number that indicates the node the user would like to work with. The new master delivers the message to all nodes. The node that receives its own node number connects itself to the terminal and the program in that node continues with the procedure that selects commands. All other nodes will wait for the selected

node to finish. Then the program in all nodes goes back to the beginning of the procedure WHICH-NODE.

If there is only one node in the network, this procedure is omitted and the program goes directly to select commands.

The CPU command GO, which is explained later, is the only command that uses the whole network and not only one node.

3.6 Bus Request

The hardware of the node board has been designed so that the IOP and the CPU processors use the same internal bus. The software has been written so that the CPU processor has the priority in the use of the bus. If the IOP processor needs access to the node memories, it must make a bus request. However, the CPU and the IOP monitor programs never need to use the bus at the same time and the bus request becomes more inportant when user programs are written.

The IOP interrupts the CPU by clearing its own INT0 which is connected to INT1 of the CPU. The IOP waits until the CPU clears the INTERRUPT-ACKN pin P2.7 after which the IOP is allowed to use the bus. When it has finished, it returns the bus to the CPU by setting INT0.

3.7 Commands for the IOP

After the master-slave table is formed and the user has determined which node he/she would like to work with, the program in the chosen node runs the procedure CHOOSE-COMMAND that receives commands from the user. The command table with the abbreviations for all commands will first appear on the screen. Right after the table, an "*"sign is displayed. This sign appears every time the IOP is expecting a command.

The IOP commands display helpful tables, change and copy memory contents and display them on the terminal screen. The pprogram will return to CHOOSE-COMMAND procedure every time a command has been performed until EX (for EXIT) is requested.

Each command is 16 bits wide. The 8 most significant bits are called HIGHCOM and the 8 least significant bits are called LOWCOM in the software. Thus, each command is identified by two letters. When both letters for a command have been entered to the terminal, the program compares them with the available commands. If a match is found the command is called. If there is no match, the user is advised to check the command and to try again.

If a command for the CPU is entered to the IOP program, the IOP program simply calls the procedure CPU which opens the CPU-terminal connection. When the connection is established, the selected command must be entered again.

3.7.1 COMMAND TABLE

The command COMMAND-TABLE or CT displays all available commands of the CPU and the IOP on the screen. Every command has a two-letter abbreviation which is used to enter the command. When the program moves to the CHOOSE-COMMAND procedure for the first time, the command table is displayed automatically on the screen. After that, CT must be entered if the table is needed again. Figure 7 shows the content of the command table.

If more commands are later created, this table is easily modified by adding a CALL PRINT statement to the appropriate place in this procedure.

```
IOP COMMANDS:
  AT=ADDRESS TABLE
  CO=COPY
  CP=CONNECT TO CPU
  CT=COMMAND TABLE
  DI=DISPLAY
  EX=EXIT FROM CHOOSE-COMMAND
  MC=CHANGE TABLE (MASTER TABLE CHANGE)
  MT=MASTER TABLE
  WM=WRITE MEMORY
CPU COMMANDS:
  CR=COMMAND REGISTER OF 8155
  LD=LOAD
  GO=START USER PROGRAM
Q=QUIT ANY COMMAND
```

Figure 7.Command table

3.7.2 ADDRESS TABLE

The ADDRESS TABLE or AT shows all the available addresses and in what kind of memories the addresses are found. By giving the command *AT, the table in figure 8 is displayed on the screen:

```
*=============================================================*
*MAIN RAM  *PORT EXP. *P.EXP.REG *EEPROM     *RAM          *
*0000-007F *9E00-9EFF *9F00-9F05 *A800-AFFF *B000-B7FF *
*=============================================================*
```

Figure 8. Address table

The MAIN RAM is the on chip RAM of the IOP or the CPU. PORT EXP is the data memory of the 8155 port expander. P.EXP. REG is the command register of the 8155. This register is discussed in more detail in the CPU program. EEPROM and RAM are the only continuous addresses from A800 to B7FF.

3.7.3 DISPLAY

The command DISPLAY or DI shows the contents of 16 addresses starting from the address that the user selects. When *DI is entered, the following text will be shown on the screen:

```
*DI
ADDRESS
```

A four digit address is expected from the user. Right after the last digit is received, the contents of 16 memory locations

will appear on the screen. For example, if the user chooses to see the contents of addresses B500-B503, the following text is displayed:

```
*DI
ADDRESS:
B500 00 11 22 33 44 55 66 77 88 99 AA BB CC DD FF
```

If another address is wanted after this, the user pushes any key except Q. When the user wants to exit from the command DISPLAY, Q (for QUIT) is entered and the program is ready to take another command.

3.7.4 WRITE MEMORY

The command WRITE MEMORY or WM gives the possibility to change a memory content of 1 to 16 addresses. The user enters a four digit address, and 16 consecutive memory contents starting from the selected address are displayed. The cursor then appears below the first memory content as follows:

```
*WM
ADDRESS:
B000  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

The new memory content must be written below the old content. The cursor then moves automatically below the next contents. If a content should remain unchanged, a space will skip that address. It is not necessary to go through all 16 addresses. When no more changes are desired, a CR will display the current

contents of all the 16 addresses. The next address (first address + 16) is then displayed automatically. If the user wants to work with that address, any key except CR and Q will display the contents of the next 16 addresses.

A CR will allow the user to select a completely new address:

```
B000   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
       11 22 33(CR)
       11 22 33 FF FF FF FF FF FF FF FF FF FF FF FF FF
B010(CR)
ADDRESS:
```

The user can now select any new address.

Q returns the program to the procedure CHOOSE-COMMAND.

3.7.5 COPY

The COPY or CO command copies up to 256 bytes to another memory location. It can copy data between main memory and a node memory, or from a node memory to another node memory, or relocate data within main memory RAM. The user must be careful when copying data to main memory RAM. The monitor program stores variables to the beginning addresses of the RAM. Normally addresses 60H to 7FH are empty and the user can write to them without causing damage to the monitor program.

The program will ask the user the starting address of the data to be copied. Next, the user is asked to enter the starting address of the destination where the data are copied. All addresses need to have four digits. When the user enters a

two digit number indicating how many data should be copied, the copying is performed. The following text is shown on the screen:

```
*CO
SOURCE:XXXX  DESTINATION:XXXX  LENGTH:XX
```

The results of the copying are not shown. The command DISPLAY can be used to check that copying was correct. When the user wants to exit from the COPY command a 'Q' is entered. If any other key is pressed, another copy command is expected.

3.7.6 MASTER TABLE

The command MASTER-TABLE or MT displays the current master-slave table that was created at the beginning of the main program and may have been changed later with the CHANGE-TABLE command. The table contains as many items as there are nodes. For example, the screen display could be as follows:

```
*MT
 00 01
 01 03
 02 04
 03 02
```

There are four nodes in use. The order of the table goes from zero to one less than the number of nodes. The order is in the left-hand column. The right-hand column indicates the node numbers which go from one to the number of nodes. In this example, node #1 is the highest master and node #2 is in the

lowest position in the table.

3.7.7 CHANGE TABLE

The master-slave table can be changed at any time. The procedure CHANGE-TABLE or MC differs from other commands in its use. It can be called, not only through the terminal, but also in user programs. If the change of table is made through the terminal, the user has to give the order that the node is seeking, before the command actually is called. If the change of the table is made through the user program, the procedure CHANGE TABLE can be called directly by passing the parameter NEW-ORDER from the user program to the called procedure.

This procedure is best explained with an example. Suppose that after the master-slave table is formed node #5 is the fifth node in the master-slave table (order=4). The user wants this node to be the third in the table (order=2). The user chooses the command MC in the node #5 and the following text will appear on the terminal:

NEW-ORDER:

02 is entered indicating the order that node #5 is seeking. The program then calls the command CHANGE-TABLE. First node #5 cleares INT1. INT1s of all nodes are connected together. When the INT1 pin of node #5 is cleared the IOPs of all nodes are interrupted. Next, node #5 disconnects itself from the terminal

and sends its own node number (5) through P1.4-7 lines to all nodes. All other nodes are now in the interrupt procedure and are looking for the P1.4-7 lines. Node #5 sets the INT1 pin. Then it sends NEW-ORDER, number 2 in this case, to all nodes through the serial line. When all nodes have received NEW-ORDER=2 and NODE-NUMBER=5, they are all ready to make the necessary changes to the table.

First the nodes save the master-slave table from the order 02 to 04 to a temporary table TEMP-TABLE. Then they write node #5 to the third place (order 02) to the master-slave table. The temporary table without the last item is written right below node #5. The last item in the TEMP-TABLE is node #5 which has already been written to its new place in the master-slave table.

Node #5 reconnects itself to the terminal and is ready to take the next command. All other nodes return from the interrupt procedure to the place where they were interrupted.

3.7.8 QUIT and EXIT

All commands can be terminated by entering Q (for QUIT) to the terminal. Q will take the program back to the CHOOSE-COMMAND procedure.

If the user wants to exit from CHOOSE-COMMAND and to select another node, EX (for EXIT) takes the program to the procedure

WHICH-COMMAND. All nodes that were not selected are waiting in this procedure for the selected node to return. The highest master takes over the terminal and the user can select another node.

3.7.9 Connect to the CPU

The IOP and the CPU processors share the same serial input and output lines RXD and TXD. In order to prevent the simultaneous use of these lines, the serial reception (REN of SCON) is disabled in one processor and enabled in the other. This is the main function of the CONNECT TO CPU or CP command.

The command CP interrupts the CPU with a bus request. The IOP then writes F0H to the address 9E20H (CPU-IOP byte) and disables its own serial reception by clearing REN.

The CPU checks the address 9E20H after each bus request. If it finds F0H in that address, it first clears the address and then enables its own serial reception. The CPU is now ready to take commands.

3.8 Procedures Common for Both the CPU and the IOP Programs

There are tasks in the program that are performed so frequently that they are written into small procedures. These procedures are placed at the beginning of the program and they

involve mainly serial inputs and outputs, changing ASCII codes to hexadecimals and vice versa. These procedures are needed in the IOP and in the CPU because both processors have access to the serial lines.

3.8.1 PUT-CHAR

PUT-CHAR is a simple procedure that outputs a byte to the serial line TXD. First the program waits for the transmit interrupt flag TI in SCON to be set. As soon as the TI=1 the serial buffer SBUF is ready to transmit a new byte. The TI bit is set automatically when the stop bit is sent out of the SBUF. The TI bit is then cleared in the software and the byte is written to the SBUF which initiates the transmission.

One must be careful when using this procedure for the node-to-node connections where the timing is very important. To make sure that the right node receives the sent information, it is recommended to write after a CALL PUT-CHAR statement the following: DO WHILE NOT TI; END;.

3.8.2 GET-CHAR

GET-CHAR is a typed procedure, which means that it can be used as any function in the program. This procedure receives a byte through the serial input RXD. The software waits for the

receive interrupt flag RI to be set. RI=1 initiates the serial reception. The RI is then cleared in the software and the byte is received from the SBUF. The NODE-FLAG bit in the procedure tells if the data transfer occurs from another node or from the terminal to the node. If the byte is received from the terminal, it is echoed to the screen.

3.8.3 PRINT

PRINT procedure outputs a string of bytes that are stored at the addresses starting from STR-P. The procedure PUT-CHAR is called in a loop that increments the address until a zero is found at the address. This procedure is meant to output ASCII codes; therefore, it is to be used only as output to the terminal.

3.8.4 ASCII-TO-HEX and HEX-TO-ASCII

The procedure ASCII-TO-HEX converts ASCII codes to hexadecimals. Two bytes must be entered at a time.

First the program checks if the first byte received is a space or a carriage return. If it is either one, the DATA-FLAG bit is cleared and the program returns this first byte to the calling program. This feature is used in the WRITE-MEMORY command as a sign to skip some memory locations.

If the first byte is a number from 0 to F, its ASCII code byte is changed to hexadecimal by substracting 30H for numbers 0 to 9 and 37H for numbers A to F. This then is signed as the four most significant bits (HIGHB) of the returned value.

The four least significant bits (LOWB) are made of the second byte entered. It is changed to a hexadecimal in the same way as the four most significant bits.

The HEX-TO-ASCII procedure converts a hexadecimal byte to a ASCII code. It simply reverses the previous process.

3.8.5 ADDRESS SPACE

ADDRESS SPACE is a procedure that takes an address and finds where the address is located. Each memory type has its own number. These numbers are shown in figure 9. The program returns a number from 0 to 4 to the calling program. This number is a useful tool and is used for example to find out if something must be written to the EEPROM.

ON CHIP RAM	0
8155 MEMORY	1
8155 COMMAND REG.	2
NODE EEPROM	3
NODE RAM	4

Figure 9. Assigned numbers for the available memory types

CHAPTER 4

THE MONITOR PROGRAM FOR THE CPU

The CPU monitor program is short because the CPU's main function is to handle user programs. The flow chart of the CPU program is shown in figure 10. The procedures that are common to both the IOP and the CPU processors have already been discussed in Chapter 3. The CPU has a command to load a user program to the external RAM and a command to run a user program. In addition to these, there is a command to program the command register and the timer of the 8155 port expander.

4.1 The Main Program

The CPU is initialized almost in the same manner as the IOP. The differences involve interrupts, serial reception and Port 1.

The external interrupt EX1 is enabled. First the EA pin is set. This makes it possible to enable interrupts separately. Then EX1 is set to enable the external interrupt (INT1).

The REN bit of SCON is cleared to disable the serial reception. The serial input is first given to the IOP processor. Only when there is a command to the CPU processor will this

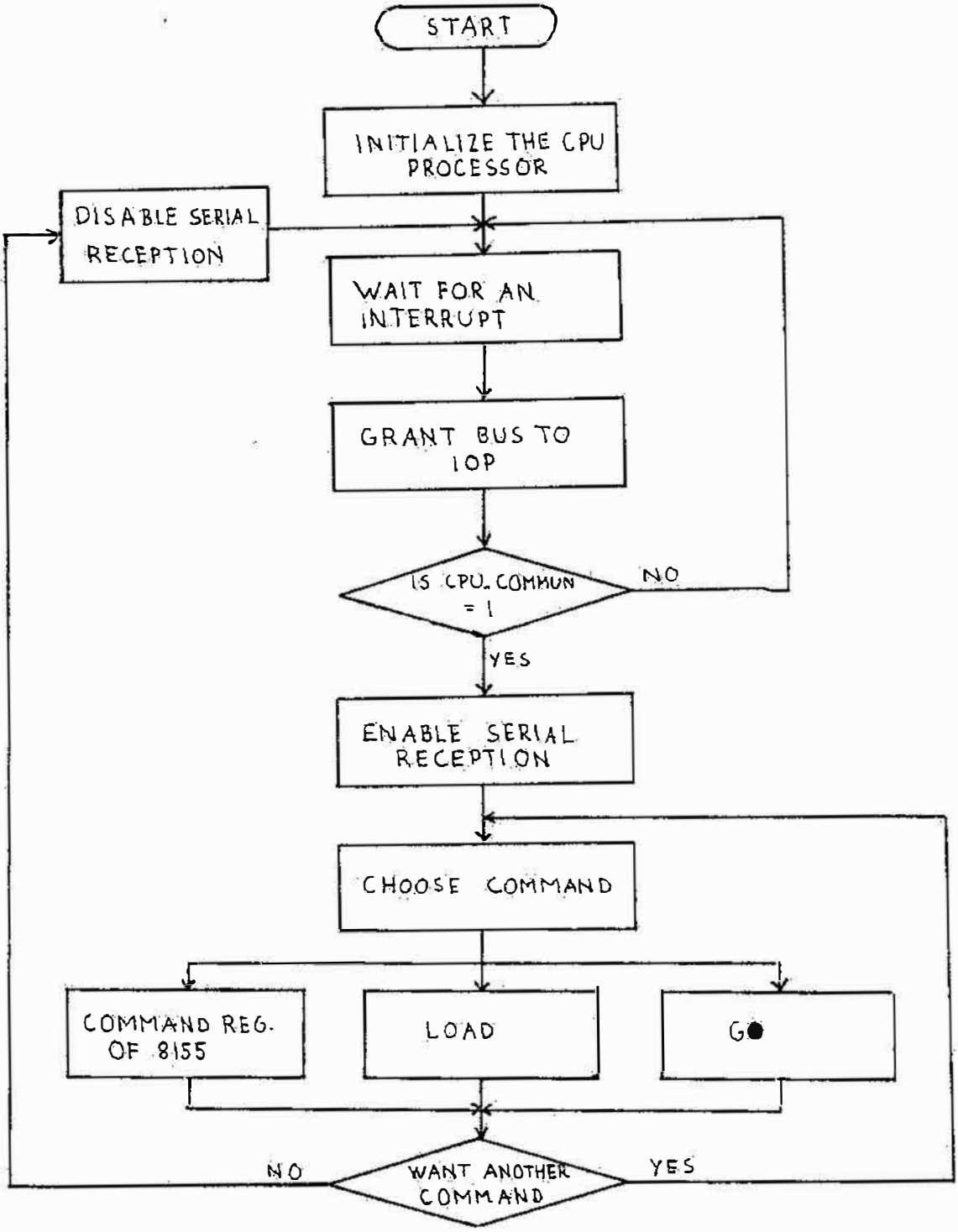


Figure 10. Flow chart of the CPU monitor program

line open by setting REN.

Right after the initialization of the processor, the program stays in a loop waiting for an interrupt. The IOP can interrupt the CPU for two reasons. It can interrupt because of a bus request, in which case, the CPU returns to wait mode. The CPU can also be interrupted to receive commands from the terminal. In this case the CPU finds F0H at the address 9E20H. The CPU-COMMUN bit is set and the program is released from wait mode. The external interrupt EX1, CPU-COMMUN and CPU-IOP bits are cleared. The screen is cleared and the text

ENTER COMMAND FOR CPU

is displayed. The program then calls the CHOOSE-COMMAND procedure.

Unlike the IOP software where the program never returns to the main program, the CPU software goes back to the main program after no more commands for the CPU are given.

4.2 Commands for the CPU

The CHOOSE-COMMAND procedure first displays a '!' in the beginning of the next row. The sign '!' tells to the user that the CPU is receiving the commands. (The IOP uses a '*'.) The CPU chooses commands in the same way as the IOP does except that there are only three commands in the command table: COMMAND REGISTER, LOAD AND GO. A new command can be selected by

pressing any key except 'Q'. 'Q' will take the program back to the IOP.

4.3 COMMAND REGISTER

The 8155 has two 8-bit wide ports A and B and a 6-bit wide port C. The ports A and B can be either inputs or outputs. The port C has four alternatives: inputs, outputs or two different combinations of control signals for Port A and Port B.

The timer is a 14-bit down-counter that counts TIMER IN pulses and provides either a square wave or pulse when the terminal count is reached. The timer can be programmed to four different modes.

The 8155 has five registers. The registers and their addresses are shown below:

COMMAND REGISTER	9F00
PORT A REGISTER	9F01
PORT B REGISTER	9F02
PORT C REGISTER	9F03
LOW-ORDER 8-BITS OF TIMER COUNT	9F04
HIGH-ORDER BITS OF TIMER COUNT	9F05

The command register of the 8155 is an 8-bit wide 'write only' register. The bit assignment of that register is shown in figure 11.

The COMMAND-REGISTER command displays the following text to the terminal:

P=PORT, T=TIMER

The user selects either P or T. If P is selected, the text in

the terminal continues:

WHICH PORT

The user selects either A,B or C.

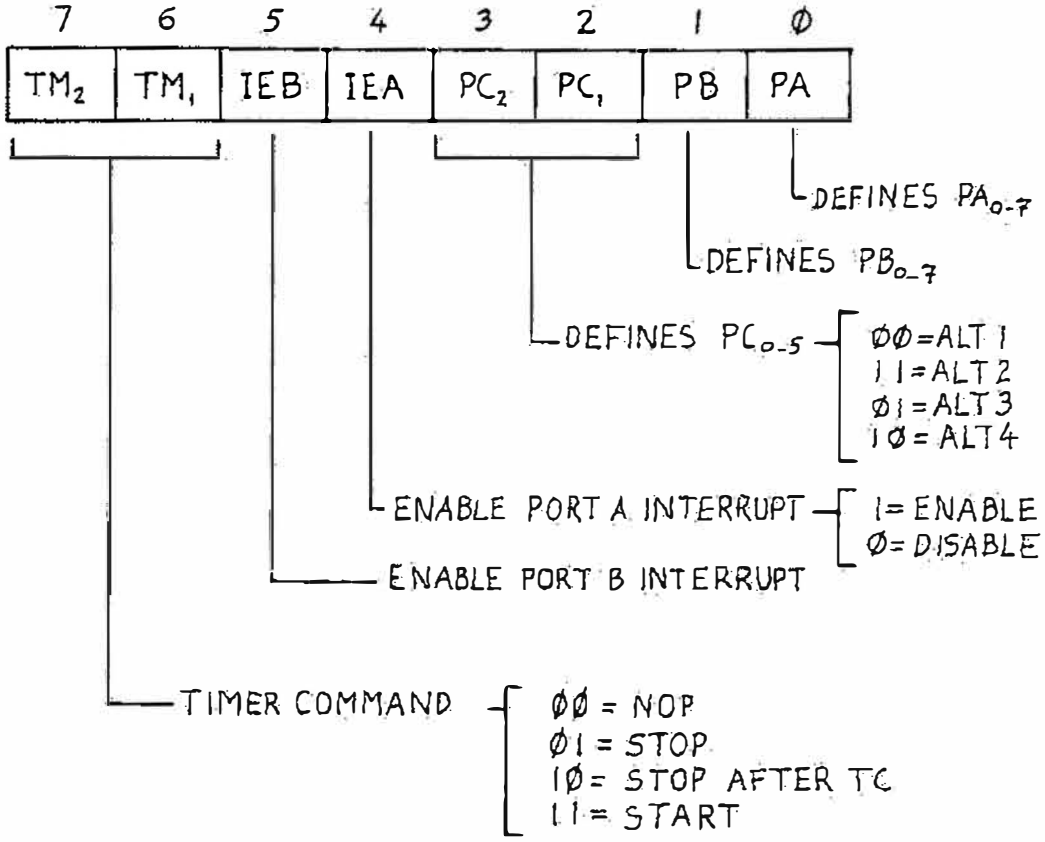


Figure 11. Bit assignment of the command register

4.3.1 Ports A, B and C

Ports A and B can be defined as inputs or outputs. The command register allows also the enabling or disabling of the interrupts of these ports (see reference 2). If the user

chooses Port A or Port B, the text on the screen asks:

I=INPUT, O=OUTPUT, R=INTERRUPT

If input or output is selected, the PB or PA bit in the command register is changed to zero for input and to one for output. Because the command register cannot be read, a copy of the register is kept in the CPU processor. All changes are always made in the command register and in the command register copy.

If Port A or B interrupt is requested, the user is asked to specify if the interrupt should be enabled or disabled. For Port A, the IEA bit of the command register is set to one to enable the interrupt and cleared to disable it. For Port B, the IEB bit of the command register is changed accordingly.

Port C can be defined as input or output or it can be used as a control for Ports A and B. Figure 12 shows four alternative modes of Port C.

ALT1 defines the whole port as input. ALT2 defines the port as output. In ALT3, the pins PC3-5 are outputs and the pins PC0-2 have control function for Port A. In ALT4, the pins PC0-2 control Port A and the pins PC3-5 control Port B.

When Port C is used as a control port, the first of the three control bits is an interrupt that the 8155 sends out. The second is an output signal indicating whether the buffer is full or empty, and the third is an input pin to accept a strobe for the strobed input mode.

PIN	ALT 1	ALT 2	ALT 3	ALT 4
PC0	INPUT PORT	OUTPUT PORT	A INTR (PORT A INTERRUPT)	A INTR (PORT A INTERRUPT)
PC1	INPUT PORT	OUTPUT PORT	A BF (PORT A BUFFER FULL)	A BF (PORT A BUFFER FULL)
PC2	INPUT PORT	OUTPUT PORT	A $\overline{\text{STB}}$ (PORT A STROBE)	A $\overline{\text{STB}}$ (PORT A STROBE
PC3	INPUT PORT	OUTPUT PORT	OUTPUT PORT	B INTR (PORT B INTERRUPT)
PC4	INPUT PORT	OUTPUT PORT	OUTPUT PORT	B BF (PORT B BUFFER FULL)
PC5	INPUT PORT	OUTPUT PORT	OUTPUT PORT	B $\overline{\text{STB}}$ (PORT B STROBE)

Figure 12. Port C control assignment

Two bits in the command register are reserved for Port C (PC1, PC2). Figure 11 shows the bit combinations for each alternative.

4.3.2 Timer

The timer is a 14-bit down-counter. The bits TM2 and TM1 in the command register are reserved for the timer. There are four commands to start and to stop the timer:

1. NOP command (TM2,TM1=00) does not affect the counter operation.
2. STOP command (TM2,TM1=01) does not affect the counter if it has not started, but the counting is stopped if the timer is running.

3. STOP AFTER TC command (TM2,TM1=10) does not affect the operation if the timer has not started. Otherwise, the timer is stopped immediately after the present terminal count is reached.
4. START command (TM2,TM1=11) loads mode and count length and starts the timer immediately after the loading if the timer is not presently running. If the timer is running, the new mode and count length are started right after the present terminal count is reached.

After the user has chosen the timer in his command, the following text is shown on the screen:

NOP=0,STOP=1,STOP AFTER TC=2,START=3

The user enters the number that corresponds to the selected operation. Then the command register and the command register copy are changed accordingly.

4.4 LOAD

If the user desires to load a program to the CPU processor, the iPDS in the ADSL is the easiest way to do it. If a program should be loaded to a node memory, the LOAD command of the CPU can do it.

The absolute file produced by relocating and linking the object file with the iPDS is not the code file that can be loaded to the node board. With the help of the Intel PROM

Programming System (iPPS), the absolute file is first loaded to the buffer where it is in the form that the 8751 microprocessor can use. This code file is copied to a disk. The serial output of the iPDS is connected to the interface board. Using iPDS, the code file is sent to the serial output. Note that it is not possible in the iPPS to send a file directly from the buffer to the serial output.

For the LOAD command, both RS232C connectors of the interface board are used.One of them is connected to the terminal as usualy. The other connector is connected to the serial output of the iPDS.

The LOAD command first displays to the terminal:

MUST FIT TO RAM B000-B7FF

This statement reminds the user to check that the program fits into the available memory space. The user then enters the starting address which must be between B000-B7FF. The question

GO? Y/N

appears next on the screen. At this point, the user should prepare the iPDS to send a code file from a disk to the serial output. When the iPDS is ready to transmit, the user enters 'Y' to the Hazeltine terminal as an answer to the GO? question.

The CPU turns off INT0 giving a signal to the IOP to clear its INT1. The INT1 of the IOP is connected through the backplane to the interface board where it receives data either from the terminal or from the iPDS. Normally, the INT1 of the

IOP is set on and the receive line of the terminal is selected. Only in the LOAD command is the INT1 of the IOP cleared and the connection from the iPDS opened.

Right after entering 'Y' to the terminal, the iPDS can start sending the code file to the network. When the file has been written to the RAM, the receive line is switched back to the terminal. The next available address is shown on the screen.

Entering 'N' instead of 'Y' to the terminal takes the software back to the CHOOSE-COMMAND procedure.

If the user wants to load a program to the EEPROM, the loading should still go into the RAM. From the RAM, the code file can be copied to the EEPROM.

4.5 GO

Even though the GO command is in the CPU, it must be selected through the IOP because this command starts the whole network. Only the IOP can send messages to other nodes.

The IOP of the selected node asks the starting address of the user program. This address is stored at 9E40H and is sent to all other nodes. Then this IOP calls the procedure CPU. Right now, the starting address is used as a sign to the CPU to call the GO command. Later, with the help of an assembly language program, the starting address will be used as an address of a user program.

All other nodes are in the GET-CHAR procedure waiting for messages from the selected node. If these nodes receive anything other than 0, they will call the CPU procedure. All nodes switch then to the CPU processor and the GO procedure is called.

The user program must be inside the processor. The first executable statement must be labeled "USER" and the program must include a RETURN statement. The user program is linked with the monitor program and then loaded into the processor (see reference 3).

CHAPTER 5

USE OF THE NETWORK AND FINAL REMARKS

Even though the monitor program of the network is written such that the user finds it easy to operate the network, there still might be some problems in setting up the system. Some assistance in starting the network is given in Section 5.1. Examples of the use of the dual processor serial network are presented in Sections 5.2 and 5.3.

5.1 Setting Up the Network

The following six steps should help the user to set up the system:

1. Install the interface board to the right-most slot. The component side must face to right.
2. Connect the interface board to the Hazeltine terminal. There are two 22 pin connectors stacked one above the other on the interface board. The lower connector is for the terminal. The other side of the cable goes to the EIA connector on the back of the Hazeltine terminal.
3. Prepare the terminal for the data transmission. See Chapter

2 for instructions for setting up the switches on the front panel. Then turn on the terminal.

4. Install the node boards (as many as you need) in the same cardcage with the interface board. Start from the left and leave no empty slots between the node boards.
5. Turn on the power in the network. A master reset may be necessary. If you have set up everything correctly

   ENTER 0 FOR CONNECTION

   is displayed on the terminal screen. The sender of this message is node #1.
6. From here on, the monitor program tells you what to do. See Chapters 3 and 4 for detailed information about the commands.

5.2 Use of a Node Board

Even using only one node board can be of significant assistance, especially for testing. A simple example is to test 8 lights. They can be connected to the 8155 Port Expander (50 pin Scotchflex connector). Suppose the lights are connected to Port A. The user chooses the CPU command COMMAND-REGISTER and defines Port A as output. The IOP command WRITE MEMORY is then applied to write to Port A (address 9F01H). The lights should turn on and off according to the data in Port A.

The node board can be used to write to a 2817 (or a pin compatible) EEPROM. The data can be written directly to the

EEPROM with the WRITE MEMORY command. The data can also be loaded to the RAM and then copied from there to the EEPROM.

These are just few examples of the use of a node board. A user can find many other applications.

5.3 Use of the Whole Network

The network is very useful in robotic applications. In fact, the dual processor serial network has already been used successfully on the ADSL robot arm. The user has to know the hardware and the software of the network as well as the hardware of the application, in order to be able to use the system.

The user is advised to start with a simple program using only one or two nodes. The node-to-node connections are not difficult to establish once the user gets used to the exact timing his/her software requires.

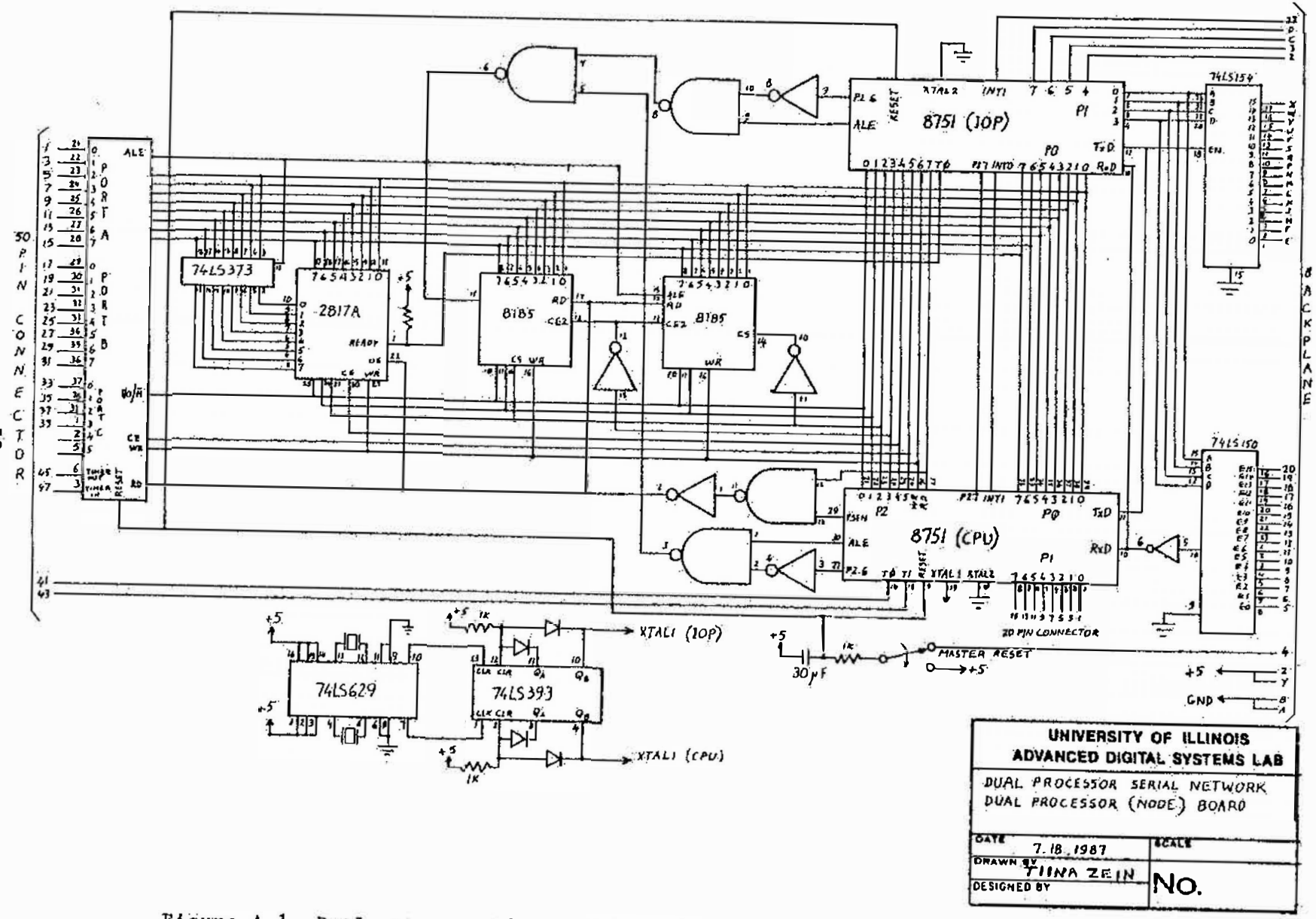
5.4 Final Remarks

The possible applications of the dual processor serial network are endless. It can be used as a teaching tool for multiprocessor systems. It opens possibilities to many projects,

especially in robotics. Finally, the network itself is an endless project. With a creative mind, it can be expanded to a network that could control a complete robot.

APPENDIX A

HARDWARE SCHEMATICS



UNIVERSITY OF ILLINOIS ADVANCED DIGITAL SYSTEMS LAB	
DUAL PROCESSOR SERIAL NETWORK DUAL PROCESSOR (NODE) BOARD	
DATE 7.18.1987	SCALE
DRAWN BY TIINA ZEIN	NO.
DESIGNED BY	

Figure A.1. Dual processor board schematic

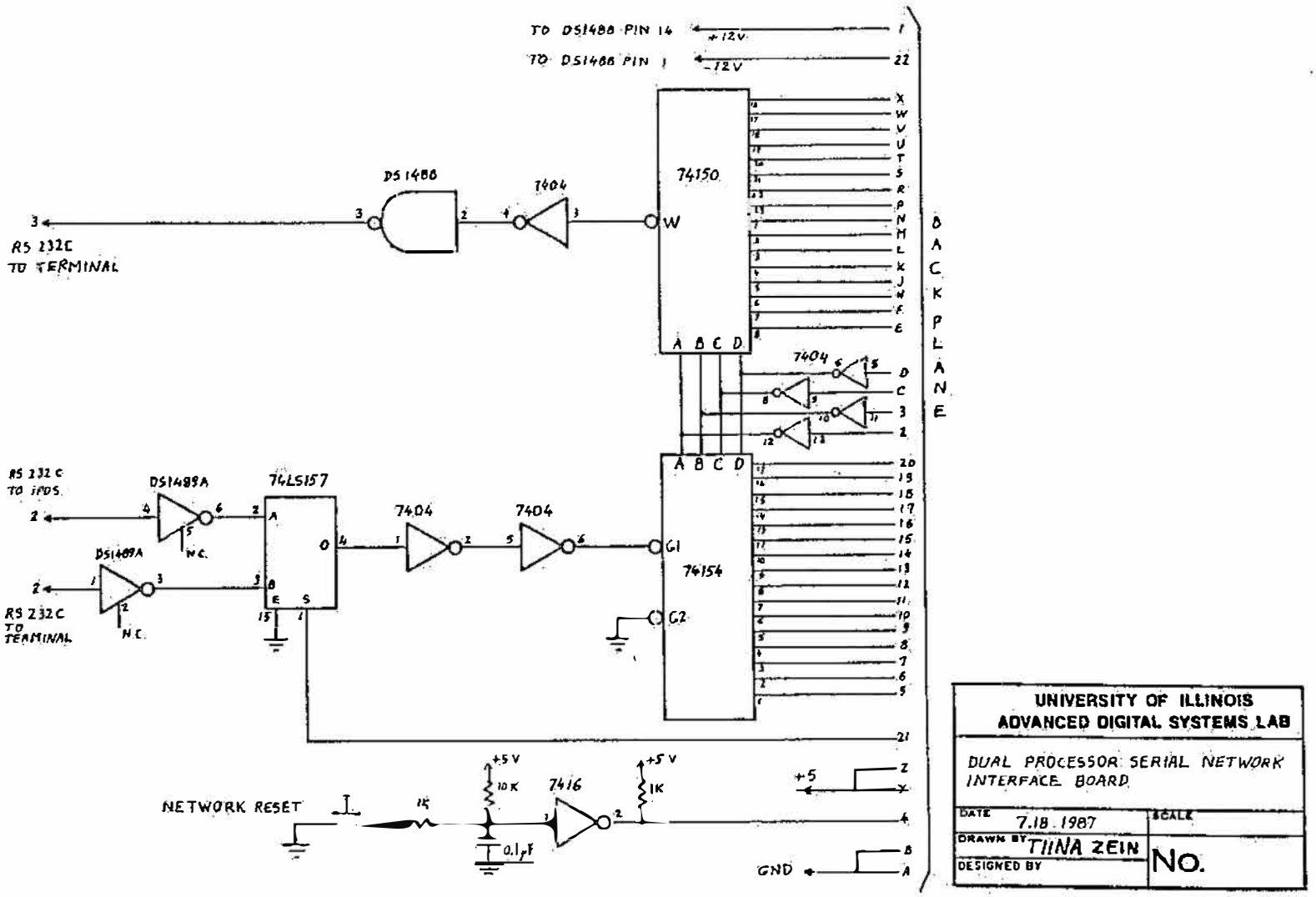


Figure A.2. Terminal interface board schematic

APPENDIX B THE IOP MONITOR PROGRAM

```
ISIS-II PL/M-51 V1.1
COMPILER INVOKED BY:  PLM51 :F1:MONIT.P51 DEBUG


                $ROM(LARGE)
                $NOCODE
  1   1         MONITOR: DO;
                $NOLIST
  5   1         DECLARE NODE BYTE;
  6   1         DECLARE CRLF LITERALLY '0DH, 0AH';
  7   1         DECLARE CR LITERALLY '0DH';
  8   1         DECLARE LF LITERALLY '0AH';
  9   1         DECLARE ECHO BYTE;
 10   1         DECLARE DATA_FLAG BIT;
 11   1         DECLARE NUMBER BYTE;
 12   1         DECLARE NEW_LOCATION WORD;
 13   1         DECLARE CONTENT_N BASED NEW_LOCATION BYTE AUXILIARY;
 14   1         DECLARE (NEW_LOCH,NEW_LOCL) BYTE AT (.NEW_LOCATION);
 15   1         DECLARE LENGTH BYTE;
 16   1         DECLARE OLD_SPACE BYTE;
 17   1         DECLARE NEW_SPACE BYTE;
 18   1         DECLARE CPU_IOP BYTE AT (09E20H) AUXILIARY;
 19   1         DECLARE EVENT BYTE;
 20   1         DECLARE LOCATN ADDRESS;
 21   1         DECLARE GOLOCH BYTE AT (9E40H) AUXILIARY;
 22   1         DECLARE GOLOCL BYTE AT (9E41H) AUXILIARY;
 23   1         DECLARE LOCATION WORD;
 24   1         DECLARE CONTENT BASED LOCATION BYTE AUXILIARY;
 25   1         DECLARE (LOCH,LOCL) BYTE AT (.LOCATION);
 26   1         DECLARE MAIN_CONTENT BASED LOCL BYTE MAIN;
 27   1         DECLARE MAIN_CONTENT_N BASED NEW_LOCL BYTE MAIN;
 28   1         DECLARE (I,J) BYTE;
 29   1         DECLARE CHAR BYTE;
 30   1         DECLARE INTERRUPT_ACKN BIT AT(0A7H) REG;
 31   1         DECLARE MY_NODE_NUMBER BYTE;
 32   1         DECLARE MY_QUARTER_NUMBER BYTE;
 33   1         DECLARE MY_VOTE1_NUMBER BYTE;
 34   1         DECLARE MY_VOTE2_NUMBER BYTE;
 35   1         DECLARE NUMBER_OF_NODES BYTE;
 36   1         DECLARE RANK BYTE;
 37   1         DECLARE MASTER_TABLE(15) BYTE AT (09E00H) AUXILIARY;
 38   1         DECLARE TEMP_TABLE(15) BYTE AUXILIARY;
 39   1         DECLARE A BYTE;
 40   1         DECLARE VOTE(2) BYTE;
 41   1         DECLARE TEMP BYTE;
 42   1         DECLARE MY_TERMINAL_NUMBER BYTE;
 43   1         DECLARE (A0,A1) BYTE;
 44   1         DECLARE ROUND BYTE;
 45   1         DECLARE (PSTART, PEND) BYTE;
 46   1         DECLARE STEP1 BYTE;
 47   1         DECLARE NODE_FLAG BIT;
 48   1         DECLARE TEMP_BIT BIT;
 49   1         DECLARE EVENT1 BYTE;
 50   1         DECLARE EVENT2 BYTE;
 51   1         DECLARE ORDER BYTE;
 52   1         DECLARE NEW_ORDER BYTE;
```

```
53  1     DECLARE ORDER_END BYTE;
54  1     DECLARE K BYTE;

          /*-------------------------------------------------------------------*/
          /*THIS INTERRUPT PROCEDURE IS USED WHEN A NODE WANTS TO GET TO A HIGHER LEVEL
            IN THE MASTER_SLAVE TABLE.*/

55  2     INTERRUPT_CHANGE:PROCEDURE INTERRUPT 2;
56  2        NUMBER = SHR(P1,4);
57  2        P1=(P1 AND 11110000B) OR NUMBER;
58  3        DO WHILE NOT RI;
59  3        END;
60  2        RI=0;
61  2        NEW_ORDER=SBUF;

62  3        DO WHILE MASTER_TABLE(ORDER)<>NUMBER;
63  3        TEMP_TABLE(ORDER) = MASTER_TABLE(ORDER);
64  3        ORDER=ORDER+1;
65  3        END;
66  2        ORDER_END = ORDER;

67  2        MASTER_TABLE(NEW_ORDER) = NUMBER;
68  3        DO ORDER = I+1 TO ORDER_END;
69  3        MASTER_TABLE(NEW_ORDER)=TEMP_TABLE(ORDER-1);
70  3        END;
71  1     END INTERRUPT_CHANGE;

          /*-------------------------------------------------------------------*/
          /*THIS PROCEDURE OUTPUTS A CHARACTER TO THE TERMINAL OR TO ANOTHER NODE.*/

72  2     PUT_CHAR: PROCEDURE(CHAR);     /*PRINT A CHAR TO SBUF*/
73  2        DECLARE CHAR BYTE;
74  3        DO WHILE NOT TI;            /*WAIT TILL READY FOR OUTPUT*/
75  3        END;
76  2        TI=0;
77  2        SBUF=CHAR;
78  1     END PUT_CHAR;

          /*-------------------------------------------------------------------*/
          /*THIS PROCEDURE RECEIVES A CHARACTER FROM THE SERIAL INPUT. IF IT COMES FROM
            THE TERMINAL IT IS ECHOED BACK.*/

79  2     GET_CHAR:PROCEDURE BYTE ;      /*GET CHAR FROM SBUF AND ECHO IT*/
80  2        DECLARE CHAR BYTE;
81  3        DO WHILE NOT RI;            /*WAIT TILL THERE IS INPUT*/
82  3        END;
83  2        RI=0;
84  2        CHAR=SBUF;
85  2        IF NOT NODE_FLAG THEN CALL PUT_CHAR(CHAR);/*ECHO IF RECEIVED FROM THE
                                                        TERMINAL*/
87  2        RETURN(CHAR);
88  1     END GET_CHAR;

          /*-------------------------------------------------------------------*/
          /*THIS PROCEDURE PRINTS A MESSAGE TO THE TERMINAL.*/

89  2     PRINT: PROCEDURE(STR_P);
```

```
 90  2      DECLARE STR_P ADDRESS; /*PRINT  NULL TERMINATED STRING RESIDING IN ROM AND
                                      POINTED AT BY STR_P*/
 91  2      DECLARE CHAR BASED STR_P BYTE CONSTANT;
 92  3      DO WHILE CHAR<>0;       /*TILL NULL TERMINATOR*/
 93  3      CALL PUT_CHAR(CHAR);
 94  3      STR_P=STR_P +1;
 95  3      END;
 96  1    END PRINT;

        /*-------------------------------------------------------------------*/
        /*THIS PROCEDURE TAKES AN ASCII CODE FROM THE SERIAL BUFFER AND CONVERTS IT TO
          A HEXADECIMAL. TWO BYTES MUST BE ENTERED*/

 97  2    ASCII_TO_HEX: PROCEDURE BYTE;
 98  2      DECLARE HIGHB BYTE;
 99  2      DECLARE LOWB BYTE;
100  2      DECLARE A BYTE;
101  2      DECLARE B BYTE;

102  2    AGAIN: DATA_FLAG=1B;
103  2      A=GET_CHAR;
104  3      IF A=20H OR A=0DH THEN DO;  /*IF SPACE OR CARRIAGE_RETURN RECEIVED RETURN*/
106  3          DATA_FLAG=0B;           /*WITH DATA_FLAG=0*/
107  3          CHAR=A;
108  3          GO TO R;
109  3      END;
110  2      IF A - 30H <= 9H THEN HIGHB = A - 30H;
112  2      ELSE HIGHB = A - 37H ;
113  2      HIGHB = SHL (HIGHB,4);

114  2      B=GET_CHAR;
115  3      IF B=20H THEN DO;
117  3          CALL PUT_CHAR(08H);     /*IF SPACE RECEIVED THEN INSERT TWO BACK-*/
118  3          CALL PUT_CHAR(08H);     /*SPACES AND GO TO AGAIN*/
119  3          GO TO AGAIN;
120  3      END;
121  2      IF B -30H <= 9H THEN LOWB = B - 30H;
123  2      ELSE LOWB = B - 37H ;
124  2      CHAR = HIGHB + LOWB ;
125  2    R: RETURN(CHAR);
126  1    END ASCII_TO_HEX;

        /*-------------------------------------------------------------------*/
        /*THIS PROCEDURE TAKES A BYTE FROM THE MEMORY AND CONVERTS IT TO A ASCII CODE
          AND SENDS IT TO SBUF*/

127  2    HEX_TO_ASCII:PROCEDURE(DATA);
128  2      DECLARE DATA BYTE;
129  2      DECLARE C BYTE;

130  2      C=SHR(DATA,4);
131  2      IF C>9H THEN C=C+37H;
133  2      ELSE C=C+30H;
134  2      CALL PUT_CHAR(C);

135  2      C=DATA AND 0FH;
136  2      IF C>9H THEN C=C+37H;
```

```
138  2        ELSE C=C+30H;
139  2        CALL PUT_CHAR(C);
140  1     END HEX_TO_ASCII;

           /*-------------------------------------------------------------------*/
           /*THIS PROCEDURE SENDS A STRING OF SPACES*/

141  2     SPACE:PROCEDURE;
142  2     I=0;
143  3     DO WHILE I<NUMBER;
144  3     CALL PUT_CHAR(20H);
145  3     I=I+1;
146  3     END;
147  1     END SPACE;

           /*-------------------------------------------------------------------*/
           /*THIS PROCEDURE REQUESTS THE BUS FROM THE CPU*/

148  2     BUS_REQ:PROCEDURE;
149  2     INT0=0;                    /*SEND A SIGN THAT THE BUS IS NEEDED*/
150  3     DO WHILE INTERRUPT_ACKN;   /*WAIT FOR THE INTERRUPT_ACKN TO CLEAR*/
151  3     END;
152  2     RETURN;
153  1     END BUS_REQ;

           /*-------------------------------------------------------------------*/
           /*THIS PROCEDURE DISPLAYS THE TYPE OF AVAILABLE MEMORIES WITH THEIR
             ADDRESSES*/

154  2     ADDRESS_TABLE:PROCEDURE;
155  2     CALL PRINT(.(CRLF, 0) );
156  2     I=1;
157  3     DO WHILE I<57D;
158  3     CALL PUT_CHAR(3DH);
159  3     I=I+1;
160  3     END;
161  2     CALL PRINT(.(CRLF,'*MAIN RAM', 0) );
162  2     NUMBER=2H;
163  2     CALL SPACE;
164  2     CALL PRINT(.('*PORT EXP.', 0) );
165  2     CALL PUT_CHAR(20H);
166  2     CALL PRINT(.('*P.EXP.REG ', 0) );
167  2     CALL PRINT(.('*EEPROM', 0) );
168  2     NUMBER=4H;
169  2     CALL SPACE;
170  2     CALL PRINT(.('*RAM', 0) );
171  2     NUMBER=7H;
172  2     CALL SPACE;
173  2     CALL PRINT(.('*', CRLF, 0) );
174  2     CALL PRINT(.('*0000-007F *9E00-9EFF *9F00-9F05 ', 0) );
175  2     CALL PRINT(.('*A800-AFFF *B000-B7FF *', CRLF, 0) );
176  2     I=1;
177  3     DO WHILE I<57D;
178  3     CALL PUT_CHAR(3DH);
179  3     I=I+1;
180  3     END;
181  2     CALL PRINT(.(CRLF, 0) );
```

```
182  2     RETURN;
183  1     END ADDRESS_TABLE;

           /*-------------------------------------------------------------------*/
           /*THIS PROCEDURE DISPLAYS A TABLE OF ALL COMMANDS WITH THEIR ABBREVIATIONS*/

184  2     COMMAND_TABLE:PROCEDURE;
185  2     CALL PRINT(.(CRLF,'COMMAND TABLE:',0) );
186  2     CALL PRINT(.(CRLF,'IOP COMMANDS:',0) );
187  2     CALL PRINT(.(CRLF,'  AT=ADDRESS TABLE',0) );
188  2     CALL PRINT(.(CRLF,'  CO=COPY',0) );
189  2     CALL PRINT(.(CRLF,'  CP=CONNECT TO CPU',0) );
190  2     CALL PRINT(.(CRLF,'  CT=COMMAND TABLE',0) );
191  2     CALL PRINT(.(CRLF,'  DI=DISPLAY',0) );
192  2     CALL PRINT(.(CRLF,'  EX=EXIT FROM THE NODE',0) );
193  2     CALL PRINT(.(CRLF,'  MC=CHANGE MASTER_TABLE',0) );
194  2     CALL PRINT(.(CRLF,'  WM=WRITE TO MEMORY',0) );
195  2     CALL PRINT(.(CRLF,'CPU COMMANDS:',0) );
196  2     CALL PRINT(.(CRLF,'  CR=COMMAND REGISTER OF 8155',0) );
197  2     CALL PRINT(.(CRLF,'  LO=LOAD EEPROM',0) );
198  2     CALL PRINT(.(CRLF,'  GO=START USER PROGRAM',0) );
199  2     RETURN;
200  1     END COMMAND_TABLE;

           /*-------------------------------------------------------------------*/
           /*THIS PROCEDURE DISPLAYS THE MASTER_SLAVE TABLE*/

201  2     MSTER_TABLE:PROCEDURE;
202  2     CALL BUS_REQ;

203  3     DO ORDER=0 TO NUMBER_OF_NODES-1;
204  3        CALL PRINT(.(CRLF,0) );
205  3        CALL HEX_TO_ASCII(ORDER);
206  3        CALL PRINT(.('   ',0) );
207  3        CALL HEX_TO_ASCII(MASTER_TABLE(ORDER));
208  3     END;
209  2     INTO=1;    /*RELEASE THE BUS*/
210  2     RETURN;
211  1     END MSTER_TABLE;
           /*-------------------------------------------------------------------*/
           /*THIS PROCEDURE SENDS A MESSAGE TO ALL NODES THAT THIS NODE WANTS TO HAVE A
             HIGHER POSITION IN THE MASTER_SLAVE TABLE. THE TABLE IS THEN CHANGED.*/

212  2     CHANGE_TABLE:PROCEDURE(NEW_ORDER);
213  2     DECLARE NEW_ORDER BYTE;

214  2        INT1=0;
215  2        P1=(P1 AND 00001111) OR SHL(MY_NODE_NUMBER,4);
216  2        ACC=0;
217  2        ACC=0;
218  2        ACC=0;
219  2        ACC=0;
220  2        ACC=0;
221  2        INT1=1;

222  2        PSTART=(P1 AND 11110000B) OR 00000001B;
223  2        PEND=(P1 AND 11110000B) OR NUMBER_OF_NODES;
```

```
224  3        DO P1 = PSTART TO PEND;
225  3          CALL PUT_CHAR(NEW_ORDER);
226  3        END;

227  2        ORDER=NEW_ORDER;
228  3        DO WHILE MASTER_TABLE(ORDER)<> MY_NODE_NUMBER;
229  3          TEMP_TABLE(ORDER)=MASTER_TABLE(ORDER);
230  3          ORDER=ORDER + 1;
231  3        END;
232  2        ORDER_END = ORDER;

233  2        MASTER_TABLE(NEW_ORDER)=MY_NODE_NUMBER;
234  3        DO ORDER = NEW_ORDER+1 TO ORDER_END;
235  3          MASTER_TABLE(ORDER)=TEMP_TABLE(ORDER-1);
236  3        END;
237  2        P1=MY_TERMINAL_NUMBER;
238  1      END CHANGE_TABLE;

           /*-------------------------------------------------------------------*/
           /*THIS PROCEDURE FINDS THE TYPE OF MEMORY WHERE AN ADDRESS IS LOCATED */

239  2      ADDRESS_SPACE:PROCEDURE(ADDRS) BYTE;
240  2      DECLARE ADDRS WORD;
241  2      DECLARE CLASS BYTE;
242  2      DECLARE TEMP BYTE;
243  2      DECLARE TEMPL BYTE;
244  2      TEMP=HIGH(ADDRS);
245  2      TEMPL=LOW(ADDRS);

246  2      CLASS=0H;
247  2      IF TEMP=00H AND TEMPL>7FH THEN CLASS=0FH;
249  2      IF TEMP>00H THEN CLASS=0FH;
251  2      IF TEMP>9DH THEN CLASS=1H;
253  2      IF TEMP>9EH THEN CLASS=2H;
255  2      IF TEMP>=9FH AND TEMPL<05H THEN CLASS=0FH;
257  2      IF TEMP>0A7H THEN CLASS=3H;
259  2      IF TEMP>0AFH THEN CLASS=4H;
261  2      IF TEMP>0B7H THEN CLASS=0FH;
263  2      RETURN(CLASS);
264  1      END ADDRESS_SPACE;

           /*-------------------------------------------------------------------*/
           /*THIS PROCEDURE DISPLAYS THE CONTENT OF 16 ADDRESSES STARTING WITH THE GIVEN
             ADDRESS AND LEAVES TWO SPACES BETWEEN DATA*/

265  2      DISPLAY:PROCEDURE;
266  2      DECLARE DATA BYTE;

267  2      LOOP:CALL PRINT (.(CRLF,'ADDRESS:',CRLF,0) );/*TAKE A NEW ADDRESS AND STORE IT*/
268  2      LOCH=ASCII_TO_HEX;                           /*AT LOCATN*/
269  2      LOCL=ASCII_TO_HEX;
270  2      LOCATN=LOCATION;
271  2      CALL PUT_CHAR(20H);
272  2      I=0;                          /*PRINT THE CONTENT OF 16 ADDRESSES LEAVING*/
273  3        DO WHILE I<16;              /*TWO SPACES BETWEEN THEM*/
274  3          CALL PRINT (.(' ', 0) );
275  3          IF LOCH=0 THEN DATA=MAIN_CONTENT;
```

```
277  4          ELSE DO;                  /*IF IN THE EXTERNAL MEMORY REQUEST THE BUS*/
278  4             CALL BUS_REQ;          /*FROM THE CPU.*/
279  4             DATA=CONTENT;
280  4             INTO=1;
281  4          END;

282  3          CALL HEX_TO_ASCII(DATA);
283  3          LOCATION=LOCATION+1;
284  3          I=I+1;
285  3       END;

286  2       CHAR=GET_CHAR;
287  2       IF CHAR<>'Q' THEN GO TO LOOP;  /*RETURN TO CHOOSE_COMMAND ONLY IF QUIT */
                                           /*ENETERED.*/
289  2       RETURN;
290  1       END DISPLAY;

          /*-------------------------------------------------------------------*/
          /*THIS PROGRAM TAKES THE ADDRESS AND SHOWS THE CONTENT OF THAT ADDRESS AND
            ALSO SHOWS THE CONTENT OF NEXT 15 ADDRESSES. CONTENTS CAN BE MODIFIED
            BY WRITING THE NEW CONTENT BELOW THE OLD ONE.*/

291  2       WRITE_MEMORY: PROCEDURE;
292  2       DECLARE DATA BYTE;
293  2       DECLARE RANK BYTE;

294  2       CALL PRINT (.(CRLF, 0) );
295  2       START:CALL PRINT (.(' ADDRESS:',CRLF,0) );/*TAKE A NEW ADDRESS AND STORE IT*/
296  2       LOCH=ASCII_TO_HEX;                        /*AT LOCATN*/
297  2       LOCL=ASCII_TO_HEX;
298  2       LOCATN=LOCATION;
299  2       CALL PUT_CHAR(20H);

300  2       CONT:I=0;                 /*PRINT THE CONTENT OF 16 ADDRESSES LEAVING*/
301  3       DO WHILE I<16;            /*TWO SPACES BETWEEN THEM*/
302  3          CALL PRINT (.('  ', 0) );
303  3          IF LOCH=0 THEN DATA=MAIN_CONTENT;

305  4          ELSE DO;               /*IF IN THE EXTERNAL MEMORY THE BUS IS */
306  4             CALL BUS_REQ;       /*NEEDED FROM THE CPU.*/
307  4             DATA=CONTENT;
308  4             INTO=1;
309  4          END;

310  3          CALL HEX_TO_ASCII(DATA);
311  3          LOCATION=LOCATION+1;
312  3          I=I+1;
313  3       END;

314  2       CALL PUT_CHAR(CR)          /*MOVE THE CURSOR BELOW THE FIRST DATA*/;
315  2       CALL PUT_CHAR(LF);
316  2       CHAR=10H;
317  2       J=0;
318  3       DO WHILE J<7;
319  3          CALL PUT_CHAR(CHAR);
320  3          J=J+1;
```

```
321  3     END;

322  2     LOCATION=LOCATN;                  /*MAKE CHANGES TO THE DATA*/
323  3     DO WHILE LOCATION<LOCATN+10H;
324  3        DATA=ASCII_TO_HEX;
325  3        IF NOT DATA_FLAG AND DATA=CR THEN GO TO X;
327  3        IF NOT DATA_FLAG AND DATA=20H THEN CALL PUT_CHAR(20H);
329  4        ELSE DO;
330  4           IF LOCH=0 THEN MAIN_CONTENT=DATA;

332  5           ELSE DO;
333  5             CALL BUS_REQ;
334  6             DO WHILE NOT TO;
335  6             END;
336  5             CONTENT=DATA;
337  5             INTO=1;
338  5           END;

339  4        END;
340  3        CALL PRINT(.(' ', 0) );
341  3        LOCATION=LOCATION+1;
342  3     END;

343  2     CALL PUT_CHAR(CR);
344  2     X:CALL PRINT(.('       ', 0) );
345  2     LOCATION=LOCATN;

346  3     DO WHILE LOCATION<LOCATN+10H     /*PRINT THE NEW CONTENTS OF 16 ADDRESSES*/;
347  3        IF LOCH=0 THEN CALL HEX_TO_ASCII(MAIN_CONTENT);

349  4        ELSE DO;
350  4           CALL BUS_REQ;
351  4           CALL HEX_TO_ASCII(CONTENT);
352  4           INTO=1;
353  4        END;

354  3        LOCATION=LOCATION+1;
355  3        CALL PRINT(.(' ', 0) );
356  3     END;

357  2     CALL PRINT(.(CRLF, 0) );          /*SHOW THE NEXT ADDRESS*/
358  2     LOCATN=LOCATION;
359  2     CALL HEX_TO_ASCII(LOCH);
360  2     CALL HEX_TO_ASCII(LOCL);
361  2     CHAR=GET_CHAR;
362  2     IF CHAR=CR THEN GO TO START;      /*CARRIAGE RETURN ENTERED IF ANOTHER */
                                           /*IS WANTED.*/
364  2     IF CHAR<>'Q' THEN GO TO CONT;     /*RETURN TO CHOOSE_COMMAND ONLY IF QUIT */
                                           /*IS ENTERED.*/
366  2     RETURN;
367  1     END WRITE_MEMORY;

        /*-------------------------------------------------------------------*/
        /*THIS PROCEDURE COPIES LENGTH NUMBER OF DATA FROM LOCATION IN THE EXTERNAL
          MEMORY TO NEW_LOCATION IN THE EXTERNAL OR MAIN MEMORY STARTING FROM
          LOCATION*/
```

```
368  2     MOVEXX:PROCEDURE;

369  2     CALL BUS_REG;
370  3     DO I=1 TO LENGTH;
371  4        DO WHILE NOT TO;
372  4        END;
373  3        IF NEW_LOC=0 THEN MAIN_CONTENT_N=CONTENT;
375  3        ELSE CONTENT_N=CONTENT;
376  3        LOCATION=LOCATION + 1;
377  3        NEW_LOCATION = NEW_LOCATION + 1;
378  3     END;
379  2     INTO=1;

380  1     END MOVEXX;

           /*-------------------------------------------------------------------*/
           /*THIS PROCEDURE COPIES LENGTH NUMBER OF DATA FROM LOCATION IN THE EXTERNAL
             MEMORY TO NEW_LOCATION IN THE EXTERNAL OR MAIN MEMORY STARTING FROM
             LOCATION + LENGTH.*/

381  2     REVERSE_MOVEXX:PROCEDURE;

382  2     LOCATION=LOCATION + LENGTH-1;
383  2     NEW_LOCATION=NEW_LOCATION+LENGTH-1;

384  2     CALL BUS_REG;
385  3     DO I=1 TO LENGTH;
386  4        DO WHILE NOT TO;
387  4        END;
388  3        IF LOC=0 THEN CONTENT_N=MAIN_CONTENT;
390  3        ELSE CONTENT_N=CONTENT;
391  3     LOCATION=LOCATION-1;
392  3     NEW_LOCATION=NEW_LOCATION-1;
393  3     END;
394  2     INTO=1;

395  1     END REVERSE_MOVEXX;

           /*-------------------------------------------------------------------*/
           /*THIS PROCEDURE COPIES LENGTH NUMBER OF DATA FROM LOCATION IN MAIN MEMORY
             TO NEW_LOCATION IN THE MAIN MEMORY STARTING FROM LOCATION.*/

396  2     MAIN_MOVEXX:PROCEDURE;

397  3     DO I=1 TO LENGTH;
398  3     MAIN_CONTENT_N=MAIN_CONTENT;
399  3     LOC=LOC + 1;
400  3     NEW_LOC = NEW_LOC + 1;
401  3     END;
402  1     END MAIN_MOVEXX;

           /*-------------------------------------------------------------------*/
           /*THIS PROCEDURE COPIES LENGTH NUMBER OF DATA FROM LOCATION IN MAIN MEMORY
             TO NEW_LOCATION IN THE MAIN MEMORY STARTING FROM LOCATION + LENGTH.*/

403  2     REVERSE_MAIN_MOVEXX:PROCEDURE;
```

```
404  2     LOCL=LOCL + LENGTH-1;
405  2     NEW_LOCL=NEW_LOCL+LENGTH-1;
406  3     DO I=1 TO LENGTH;
407  3     MAIN_CONTENT_N=MAIN_CONTENT;
408  3     LOCL=LOCL-1;
409  3     NEW_LOCL=NEW_LOCL-1;
410  3     END;
411  1     END REVERSE_MAIN_MOVEXX;

           /*-------------------------------------------------------------------*/
           /*THIS PROCEDURE COPIES ANY SIZE OF BLOCK TO A NEW LOCATION.*/

412  2     COPY:PROCEDURE;

413  2     MORE:CALL PRINT(.(CRLF,'SOURCE:', 0) );
414  2     LOCH=ASCII_TO_HEX;            /*GET THE OLD AND NEW ADDRESS AND THE LENGTH */
415  2     LOCL=ASCII_TO_HEX;            /*OF THE BLOCK TO BE COPIED*/
416  2     CALL PRINT(.(' DESTINATION:', 0) );
417  2     NEW_LOCH=ASCII_TO_HEX;
418  2     NEW_LOCL=ASCII_TO_HEX;
419  2     CALL PRINT(.(' LENGTH:', 0) );
420  2     LENGTH=ASCII_TO_HEX;

421  2     OLD_SPACE=ADDRESS_SPACE(LOCATION);
422  2     NEW_SPACE=ADDRESS_SPACE(NEW_LOCATION);
423  2     CALL HEX_TO_ASCII(OLD_SPACE);
424  2     CALL HEX_TO_ASCII(NEW_SPACE);

425  3     IF OLD_SPACE=0 AND NEW_SPACE=0 THEN DO;
427  3      IF LOCATION<NEW_LOCATION THEN CALL REVERSE_MAIN_MOVEXX;
429  3      ELSE CALL MAIN_MOVEXX;
430  3     END;

431  3     ELSE DO;
432  3         IF LOCATION<NEW_LOCATION THEN CALL REVERSE_MOVEIX;
434  3         ELSE CALL MOVEIX;
435  3     END;

436  2     CHAR=GET_CHAR;
437  2     IF CHAR<>'Q' THEN GO TO MORE;

439  2     RETURN;
440  1     END COPY;

           /*-------------------------------------------------------------------*/
           /*THIS PROCEDURE GIVES THE SERIAL LINES TO THE CPU. THE IOP DISABLES ITS
             SERIAL RECEPTION BY CLEARING REN UNTIL IT RECEIVES ZERO ON INTERRUPT_ACKN
             PIN.*/

441  2     CPU:PROCEDURE;
442  2     CALL BUS_REQ;

443  2     CPU_IOP=OFOH;   /*GIVE THE SIGN TO THE CPU TO TAKE OVER THE SERIAL LINES*/
444  2     REN=0;          /*DISABLE SERIAL RECEPTION*/
445  2     INTO=1;

446  3     DO WHILE INTERRUPT_ACKN;  /*WAIT UNTIL THE CPU IS DONE*/
```

```
447  3        IF INT0=0 THEN INT1=0;
449  3      END;
450  2      REN=1;                    /*ENABLE THE SERIAL RECEPTION*/
451  2      RETURN;
452  1      END CPU;
            /*-------------------------------------------------------------------*/
            /*THIS PROCEDURE STARTS A USER PROGRAM IN THE CPU */

458  2      USER_GO:PROCEDURE;
454  2      CALL PRINT(.('STARTING ADDRESS: ',0));

455  2      CALL BUS_REG;
456  2      GOLOCH=ASCII_TO_HEX;
457  2      GOLOCL=ASCII_TO_HEX;
458  2      INT0=1;

459  3      DO I=1 TO NUMBER_OF_NODES;
460  3        P1=(P1 AND 11110000B) OR I;
461  3        CALL PUT_CHAR(GOLOCH);
462  3        CALL PUT_CHAR(GOLOCL);
463  4        DO WHILE NOT TI;
464  4        END;
465  3        CALL CPU;
466  3      END;
467  1      END USER_GO;


            /*-------------------------------------------------------------------*/
            /*THIS PROCEDURE DISPLAYS ALL THE AVAILABLE COMMANDS. THE USER SELECTS  THE
              DESIRED COMMAND AND THE PROGRAM WILL CONTINUE WITH THAT COMMAND PROCEDURE.*/

468  2      CHOOSE_COMMAND:PROCEDURE;
469  2      DECLARE COMMAND WORD;
470  2      DECLARE (HIGHCOM, LOWCOM) BYTE AT (.COMMAND);
471  2      DECLARE COMMANDS(12) WORD CONSTANT('WM','AT','CO','MC','DI','EX','CT','MT','GO','CP','CR','LD');
472  2      DECLARE ORDER BYTE;

473  2      CALL COMMAND_TABLE;             /*DISPLAY ALL COMMANDS.*/
474  2      CALL PRINT(.(CRLF,0) );

475  2      EVER_LOOP:CALL PRINT(.('*', 0) );
476  2      HIGHCOM=GET_CHAR;          /*RECEIVE A TWO LETTER COMMAND.*/
477  2      LOWCOM=GET_CHAR;

478  2      ORDER=0;                   /*MATCH THE COMMAND WITH COMMAND TABLE.*/
479  3      DO WHILE COMMAND <> COMMANDS(ORDER);
480  3      ORDER=ORDER+1;
481  4        IF ORDER=0CH THEN DO;
483  4          CALL PRINT(.(CRLF,'CHECK COMMAND', 0));;
485  4          GO TO BBB;
486  4        END;
487  3      END;

488  2      IF ORDER>9H THEN ORDER=9H; /*ALL CPU COMMANDS CALL CPU.*/

490  3      CHOOSE:DO CASE ORDER;
491  3        CALL WRITE_MEMORY;
```

```
492  3        CALL ADDRESS_TABLE;
493  3        CALL COPY;
494  4        DO;
495  4          CALL PRINT(.(CRLF,'NEW ORDER: ',0) );
496  4          NEW_ORDER=ASCII_TO_HEX;
497  4          CALL CHANGE_TABLE(NEW_ORDER);
498  4        END;
499  3        CALL DISPLAY;
500  3        GO TO AS;
501  3        CALL COMMAND_TABLE;
502  3        CALL MSTER_TABLE;
503  3        CALL USER_GO;
504  3        CALL CPU;
505  3      END;

506  2      BB8:CALL PRINT(.(CRLF, 0) );
507  2      GO TO EVER_LOOP;
508  2      AS:PSTART=(P1 AND 11110000B) OR 00000001B;
509  2         PEND=(P1 AND 11110000B) OR NUMBER_OF_NODES;
510  3         DO P1=PSTART TO PEND;
511  3         CALL PUT_CHAR(0);
512  3         CALL PUT_CHAR(0);
513  3         END;
514  2      RETURN;
515  1      END CHOOSE_COMMAND;

         /*------------------------------------------------------------------------*/
         /* THIS PROCEDURE FINDS OUT WHICH NODE THE USER WANTS TO OPERATE.*/

516  2      WHICH_NODE: PROCEDURE;
517  2      DECLARE NODE_NUMBER BYTE;
518  2      DECLARE TRUE LITERALLY '1';

519  3      DO WHILE TRUE;

520  4      IF NUMBER_OF_NODES=1 THEN DO;
522  4         P1=MY_TERMINAL_NUMBER;
523  4         CALL CHOOSE_COMMAND;
524  4         GO TO XXX;
525  4      END;

526  4      IF MY_NODE_NUMBER=MSTER_TABLE(0) THEN DO;
528  4         P1=MY_TERMINAL_NUMBER;
529  4         CALL PRINT(.(CRLF,'WHICH NODE: ',0) );
530  4         NODE_NUMBER=ASCII_TO_HEX;
531  4         CHAR = NODE_NUMBER;

532  4         PSTART=(P1 AND 11110000B) OR 00000001B;
533  4         PEND=(P1 AND 11110000B) OR NUMBER_OF_NODES;
534  5         DO K=PSTART TO PEND;
535  5           P1=K;
536  5           CALL PUT_CHAR(CHAR);
537  6           DO WHILE NOT TI;
538  6           END;
539  5         END;

540  4      P1=MY_TERMINAL_NUMBER;
```

```
541  4     END;

542  4     ELSE DO;
543  4        P1=(P1 AND 11110000B) OR MASTER_TABLE(0);
544  4        NODE_FLAG=1;
545  4        CHAR=GET_CHAR;
546  4        NODE_FLAG=0;
547  4     END;


549  4     IF CHAR=MY_NODE_NUMBER THEN DO;
550  4        P1=MY_TERMINAL_NUMBER;
551  4        CALL CHOOSE_COMMAND;
552  4     END;

553  4     ELSE DO;
554  4        P1=(P1 AND 11110000B) OR CHAR;
555  4        CALL BUS_REG;
556  4        GOLOCH=GET_CHAR;
557  4        GOLOCL = GET_CHAR;
558  4        IF GOLOCH<>0 AND GOLOCL <> 0 THEN CALL CPU;
560  4        INTO=1;
561  4     END;

562  3     XXX:END;
563  1     END WHICH_NODE;

           /*************************************************************************/
           /*                        THE MAIN PROGRAM                               */
           /*************************************************************************/

564  2     INIT:DO; /*INITIALIZE THE PROCESSOR*/
565  2          TH1=11111110B; /*INSERT -2 TO TH1*/
566  2          SCON=01010010B;/* THE SERIAL CONTROL INITIALIZED TO 8-BIT MODE1*/
567  2          TMOD=00100000B;/*TIMER1 INITIALIZED TO 8_BIT AUTORELOAD MODE*/
568  2          ES=0;/*DISABLE THE SERIAL INTERRUPT*/
569  2          TR1=1;
570  2          NODE_FLAG=0;
571  1     END INIT;


572  2     IDENTIFY: DO;
573  2        P1=0F1H;                    /*TO FIND ITS OWN NODE NUMBER THE PROCESSOR
                                           CHECKS THE RECEIVE LINES. ITS OWN RECEIVE*/
574  3        DO WHILE RXD;               /*LINE IS CONNECTED TO GROUND.*/
575  3        P1=P1+1;
576  3        END;

577  2     MY_NODE_NUMBER=P1 AND 00001111B;

578  2     NUMBER=SHR(MY_NODE_NUMBER-1,2);
579  2     MY_VOTE1_NUMBER=4+NUMBER;
580  2     MY_VOTE2_NUMBER=4+(MY_NODE_NUMBER-1 AND 0011B);
581  2     MY_TERMINAL_NUMBER=SHL(NOT(MY_NODE_NUMBER-1),4);

582  3     IF MY_NODE_NUMBER=1 THEN DO;    /*IF I AM THE NODE #1, I AM MOMENTARILY */
                                           /* THE MASTER AND I WILL CONNECT MYSELF TO*/
```

```
                                        /* THE TERMINAL*/
584  3          P1=MY_TERMINAL_NUMBER;
585  3          CALL PRINT(.(126,1CH,0) ); /*CLEAR THE SCREEN AND PRINT A MESSAGE*/
586  3          CALL PRINT(.(CRLF,0) );
587  3          CALL PRINT(.('ENTER 0 FOR CONNECTION: ', 0) );

588  4          DO WHILE NOT TI;          /*THE USER PROVIDES THE NUMBER OF NODES IN */
589  4          END;                      /*USE AND THE NODE #1 SENDS IT TO ALL NODES*/

590  3           ECHO=GET_CHAR;
591  3           CALL PRINT(.(CRLF,'HOW MANY NODES?  ',0) );
592  3           NUMBER_OF_NODES=ASCII_TO_HEX;

593  4           DO I=1 TO 50;
594  4              CALL TIME(200);
595  4           END;

596  3           PSTART=11110001B;
597  3           PEND=11110000B OR NUMBER_OF_NODES;

598  4           DO K=PSTART TO PEND;
599  4           P1=K;
600  4           CALL PUT_CHAR(NUMBER_OF_NODES);
601  5           DO WHILE NOT TI;
602  5           END;
603  4           END;
604  3      P1=11110000B;
605  3      END;

606  3      ELSE DO;                  /*NODES OTHER THAN NODE #1  RECEIVES THE NUMBER
                                         OF NODES IN USE FROM THE NODE #1*/
607  3        P1=11110001B;
608  3        NODE_FLAG=1;
609  3        NUMBER_OF_NODES=GET_CHAR;
610  3        NODE_FLAG=0;
611  3      END;

612  1      END IDENTIFY;


613  2      MASTER_SLAVE:DO;
614  2      RANK=0;
615  2      DATA_FLAG=1;
616  2      ROUND=0;


617  3      DO WHILE RANK<NUMBER_OF_NODES;
618  4        IF DATA_FLAG THEN DO;
620  4           VOTE(ROUND)=SHR(P1,4);

621  5           IF VOTE(ROUND)=1111B THEN DO;            /*IF NOBODY SENT A ZERO [ TRY*/
623  5             IF ROUND=0 THEN P1=ROL(11111110B,MY_VOTE1_NUMBER);
625  5             ELSE P1=ROL(11111110B,MY_VOTE2_NUMBER);
626  5           END;
627  4        END;

628  4        IF MY_NODE_NUMBER=1 THEN DO;
```

```
630  4          VOTE(ROUND)=SHR(P1,4);
631  5            DO WHILE VOTE(ROUND)=1111B;
632  5              VOTE(ROUND)=SHR(P1,4);
633  5            END;

634  4          I=0;
635  4          TEMP_BIT=1;

636  5          DO WHILE TEMP_BIT;
637  5            TEMP_BIT=BOOLEAN(VOTE(ROUND));
638  5            I=I+1;
639  5            VOTE(ROUND)=ROR(VOTE(ROUND),1);
640  5          END;

641  5          IF ROUND=0 THEN DO;
643  5                A0=I-1;
644  5                I=I+3;
645  5          END;
646  5          ELSE DO;
647  5            MASTER_TABLE(RANK)=4*A0+I;
648  5            P1=11110000B;
649  5            CALL HEX_TO_ASCII(MASTER_TABLE(RANK));
650  6              DO  WHILE NOT TI;
651  6              END;
652  5          END;

653  5            DO K=11110001B TO 11110000B OR NUMBER_OF_NODES;  /* THE NODE #1 SEND A ZERO TO EACH*/
654  5              P1=K;                                          /* NODE, WAITS FOR ANY ANSWER AND*/
655  5              IF ROUND=0 THEN CALL PUT_CHAR(I);
657  5              ELSE CALL PUT_CHAR(MASTER_TABLE(RANK));
659  6                DO WHILE NOT TI;
659  6                END;
660  5             END;
661  4        P1=11110000B;
662  4        END;


663  4        ELSE DO;
664  4           NODE_FLAG=1;

665  4           VOTE(ROUND)=SHR(P1,4);
666  5           IF NOT DATA_FLAG THEN DO;
668  6           DO WHILE VOTE(ROUND)=1111B;
669  6           VOTE(ROUND)=SHR(P1,4);
670  6           END;
671  5           END;

672  4           P1=11110001B;
673  4           IF ROUND=0 THEN I=GET_CHAR;
675  4           ELSE MASTER_TABLE(RANK)=GET_CHAR;
676  4           NODE_FLAG=0;
677  4        END;


678  3        IF ROUND=0 AND I<>MY_VOTE1_NUMBER THEN DATA_FLAG=0;

680  4        IF ROUND=1 THEN DO;
```

```
682  4          DATA_FLAG=1;
683  5          DO J=0 TO RANK;
684  5            IF MASTER_TABLE(RANK)=MY_NODE_NUMBER THEN DATA_FLAG=0;
686  5          END;
687  4          RANK=RANK+I;
688  4        END;

689  3        ROUND=ROUND+1;
690  3        IF ROUND=2 THEN ROUND=0;
692  3      END;

693  3      IF MASTER_TABLE(0)=MY_NODE_NUMBER THEN DO;
695  3        P1=MY_TERMINAL_NUMBER;
696  3        CALL PRINT(.(CRLF,'MASTER:NODE #',0));
697  3        CALL HEX_TO_ASCII(MASTER_TABLE(0));
698  4          DO WHILE TI;
699  3          END;;
701  3        END;
702  2      CALL WHICH_NODE;
703  1      END MASTER_SLAVE;

           /*************************************************************************/
           /*                        END OF THE MAIN PROGRAM                        */
           /*************************************************************************/

704  1      END MONITOR;
```

```
MODULE INFORMATION:              (STATIC+OVERLAYABLE)
   CODE SIZE                     = 0C3EH      3134D
   CONSTANT SIZE                 = 0299H       665D
   DIRECT VARIABLE SIZE          =   27H+0CH    39D+ 12D
   INDIRECT VARIABLE SIZE        =   00H+00H     0D+  0D
   BIT SIZE                      =   03H+00H     3D+  0D
   BIT-ADDRESSABLE SIZE          =   00H+00H     0D+  0D
   AUXILIARY VARIABLE SIZE       = 000FH        15D
   MAXIMUM STACK SIZE            = 0011H        17D
   REGISTER-BANK(S) USED:          0
   979 LINES READ
   0 PROGRAM ERROR(S)
END OF PL/M-51 COMPILATION
```

APPENDIX C THE CPU MONITOR PROGRAM

```
ISIS-II PL/M-51 V1.1
COMPILER INVOKED BY:  PLM51 CPU1.P51 DEBUG


  1   1     MONITOR: DO;
              $NOLIST
  5   1     DECLARE NODE BYTE;
  6   1     DECLARE CRLF LITERALLY '0DH, 0AH';
  7   1     DECLARE CR LITERALLY '0DH';
  8   1     DECLARE LF LITERALLY '0AH';
  9   1     DECLARE ECHO BYTE;
 10   1     DECLARE (DATA_FLAG,CPU_COMMUN) BIT;
 11   1     DECLARE NUMBER BYTE;
 12   1     DECLARE LOCATION WORD;
 13   1     DECLARE NEW_LOCATION WORD;
 14   1     DECLARE (LOCH, LOCL) BYTE AT (.LOCATION);
 15   1     DECLARE GOLOCATION WORD AT (9E40H) AUXILIARY;
 16   1     DECLARE (NEW_LOCH,NEW_LOCL) BYTE AT (.NEW_LOCATION);
 17   1     DECLARE CONTENT BASED LOCATION BYTE AUXILIARY;
 18   1     DECLARE LENGTH BYTE;
 19   1     DECLARE OLD_SPACE BYTE;
 20   1     DECLARE NEW_SPACE BYTE;
 21   1     DECLARE CPU_IOP BYTE AT (9E20H) AUXILIARY;
 22   1     DECLARE EVENT BYTE;
 23   1     DECLARE COMMAND_REGISTER_COPY BYTE;
 24   1     DECLARE COMMAND_REGISTER BYTE AT (9F00H) AUXILIARY;
 25   1     DECLARE STATUS BYTE;
 26   1     DECLARE INTERRUPT_ACKN BIT AT (0A7H) REG;
 27   1     DECLARE NODE_FLAG BIT;
 28   1     DECLARE J BYTE;
 29   1     DECLARE USER LABEL EXTERNAL;

              /*-------------------------------------------------------------------*/
              /*THIS PROCEDURE WILL GIVE THE BUS TO THE IOP.*/

 30   2     INTERRUPT_2:PROCEDURE INTERRUPT 2;
 31   2     P0,P2=0FFH;
 32   2     INTERRUPT_ACKN=0;
 33   2     ACC=0;
 34   2     ACC=0;
 35   2     ACC=0;
 36   2     ACC=0;
 37   2     ACC=0;
 38   2     INTERRUPT_ACKN=1;
 39   3     DO WHILE INT1=0;
 40   3     END;
 41   2     IF CPU_IOP=0F0H THEN CPU_COMMUN=1;
 43   1     END INTERRUPT_2;

              /*-------------------------------------------------------------------*/
              /*THIS PROCEDURE OUTPUTS A CHARACTER TO THE TERMINAL TO ANOTHER NODE.*/

 44   2     PUT_CHAR: PROCEDURE(CHAR); /*PRINT AA CHAR TO SBUF*/
 45   2       DECLARE CHAR BYTE;
 46   3       DO WHILE NOT TI; /*WAIT TILL READY FOR OUTPUT*/
 47   3       END;
```

```
48  2       TI=0;
49  2       SBUF=CHAR;
50  1     END PUT_CHAR;

        /*-------------------------------------------------------------------*/
        /*THIS PROCEDURE RECEIVES A CHARACTER FROM THE SERIAL INPUT. IF IT COMES FROM
          THE TERMINAL IT IS ECHOED BACK.*/

51  2     GET_CHAR:PROCEDURE BYTE ;/*GET CHAR FROM SBUF AND ECHO IT*/
52  2       DECLARE CHAR BYTE;
53  3       DO WHILE NOT RI;/*WAIT TILL THERE IS INPUT*/
54  3       END;
55  2       RI=0;
56  2       CHAR=SBUF;
57  2       IF NOT MODE_FLAG THEN CALL PUT_CHAR(CHAR);
59  2       RETURN(CHAR);
60  1     END GET_CHAR;
        /*-------------------------------------------------------------------*/
        /*THIS PROCEDURE PRINTS A MESSAGE TO THE TERMINAL.*/

61  2     PRINT: PROCEDURE(STR_P);
62  2       DECLARE STR_P ADDRESS; /*PRINT  NULL TERMINATED STRING RESIDING IN ROM AND
                                    POINTED AT BY STR_P*/
63  2       DECLARE CHAR BASED STR_P BYTE CONSTANT;
64  3       DO WHILE CHAR<>0; /*TILL NULL TERMINATOR*/
65  3       CALL PUT_CHAR(CHAR);
66  3       STR_P=STR_P +1;
67  3       END;
68  1     END PRINT;

        /*-------------------------------------------------------------------*/
        /*THIS PROCEDURE TAKES AN ASCII CODE FROM THE SERIAL BUFFER AND CONVERTS IT TO
          A HEXADECIMAL. TWO BYTES MUST BE ENTERED*/

69  2     ASCII_TO_HEX: PROCEDURE BYTE;
70  2     DECLARE CHAR BYTE;
        /* THIS PROGRAM TAKES AN ASCII CODE FROM THE SERIAL BUFFER AND CONVERTS */
        /* IT TO A HEXADESIMAL */
71  2       DECLARE HIGHB BYTE;
72  2       DECLARE LOWB BYTE;
73  2       DECLARE A BYTE;
74  2       DECLARE B BYTE;

75  2     AGAIN: DATA_FLAG=1B;
76  2       A=GET_CHAR;
77  3       IF A=20H OR A=0DH THEN DO;
79  3           DATA_FLAG=0B;
80  3           CHAR=A;
81  3           GO TO R;
82  3       END;
83  2       IF A - 30H <= 9H THEN HIGHB = A - 30H;
85  2       ELSE HIGHB = A - 37H ;
86  2       HIGHB = SHL (HIGHB,4);

87  2       B=GET_CHAR;
88  3       IF B=20H THEN DO;
90  3           CALL PUT_CHAR(08H);
```

```
 91  3          CALL PUT_CHAR(03H);
 92  3          GO TO AGAIN;
 93  3        END;
 94  2        IF B -30H <= 9H THEN LOWB = B - 30H;
 96  2        ELSE LOWB = B - 37H ;
 97  2        CHAR = HIGHB + LOWB ;
 98  2      R: RETURN(CHAR);
 99  1      END ASCII_TO_HEX;

          /*-------------------------------------------------------------------*/
          /*THIS PROCEDURE TAKES A BYTE FROM THE MEMORY AND CONVERTS IT TO ASCII CODE
            AND SENDS IT TO SBUF.*/

100  2      HEX_TO_ASCII:PROCEDURE(DATA);
101  2        DECLARE DATA BYTE;
102  2        DECLARE C BYTE;

103  2        C=SHR(DATA,4);
104  2        IF C>9H THEN C=C+37H;
105  2        ELSE C=C+30H;
107  2        CALL PUT_CHAR(C);

108  2        C=DATA AND 0FH;
109  2        IF C>9H THEN C=C+37H;
111  2        ELSE C=C+30H;
112  2        CALL PUT_CHAR(C);
113  1      END HEX_TO_ASCII;

          /*-------------------------------------------------------------------*/
          /*THIS PROCEDURE FINDS THE TYPE OF THE MEMORY WHERE AN ADDRESS IS LOCATED.*/

114  2      ADDRESS_SPACE:PROCEDURE(ADDRS) BYTE;
115  2      DECLARE ADDRS WORD;
116  2      DECLARE CLASS BYTE;
117  2      DECLARE TEMP BYTE;
118  2      DECLARE TEMPL BYTE;
119  2      TEMP=HIGH(ADDRS);
120  2      TEMPL=LOW(ADDRS);

121  2      CLASS=0H;
122  2      IF TEMP=00H AND TEMPL>7FH THEN CLASS=0FH;
124  2      IF TEMP>00H THEN CLASS=0FH;
126  2      IF TEMP>97H THEN CLASS=1H;
128  2      IF TEMP>9EH THEN CLASS=2H;
130  2      IF TEMP=9FH AND TEMPL>05H THEN CLASS=0FH;
132  2      IF TEMP>0A7H THEN CLASS=3H;
134  2      IF TEMP>0AFH THEN CLASS=4H;
136  2      IF TEMP>0B7H THEN CLASS=0FH;
138  2      RETURN(CLASS);
139  1      END ADDRESS_SPACE;

          /*-------------------------------------------------------------------*/
          /*THIS PROCEDURE STARTS THE USER PROGRAM;*/

140  2      USER_GO:PROCEDURE;
141  2      DECLARE X BASED SP BYTE;
```

```
142  2     SP=SP+1;
143  2     X=0ACH;
144  2     SP=SP+1;
145  2     X=01H;

146  2     GO TO USER;

147  2     RETURN;
148  1     END USER_GO;

           /*-------------------------------------------------------------------*/
           /*THIS PROCEDURE LOADS A USER PROGRAM FROM A DISK TO EXTERNAL RAM.*/

149  2     LOAD:PROCEDURE;
150  2     CALL PRINT(.(CRLF,'MUST FIT TO RAM 8000-87FF', 0) );
151  2     CALL PRINT(.(CRLF,'STARTING ADDRESS: ', 0) );
152  2     LOCH=ASCII_TO_HEX;
153  2     LOCL=ASCII_TO_HEX;

154  2     CALL PRINT(.(CRLF,'GO? Y/N', 0) );
155  2     ECHO=GET_CHAR;
156  3     IF ECHO='Y' THEN DO;
158  3        INT1=0;
159  4        DO WHILE LOCATION<0BB00H;
160  4           J=0;
161  5           DO WHILE NOT RI OR J<200;
162  5              J=J+1;
163  5           END;
164  4           IF J=200 THEN GO TO AA;
166  5           ELSE DO;
167  5                RI=0;
168  5                CONTENT=SBUF;
169  5                LOCATION=LOCATION+1;
170  5           END;
171  4        END;
172  3     AA:INT1=1;
173  3     CALL PRINT(.('NEXT ADDRESS: ',0));
174  3     CALL HEX_TO_ASCII(LOCH);
175  3     CALL HEX_TO_ASCII(LOCL);
176  3     END;

177  2     RETURN;
178  1     END LOAD;

           /*-------------------------------------------------------------------*/
           /*THIS PROCEDURE DEFINES THE PORT A OF 8155 TO INPUT IF STATUS = I AND TO
             OUTPUT IF STATUS = 0.*/

179  2     PORT_A:PROCEDURE(STATUS);
180  2     DECLARE STATUS BYTE;

181  2     IF STATUS ='I' THEN COMMAND_REGISTER_COPY=COMMAND_REGISTER_COPY AND 11111110B;
183  2     IF STATUS ='O' THEN COMMAND_REGISTER_COPY=COMMAND_REGISTER_COPY OR 00000001B;
185  2     COMMAND_REGISTER=COMMAND_REGISTER_COPY;
186  2     RETURN;
187  1     END PORT_A;
```

```
                /*-------------------------------------------------------------------*/
                /*THIS PROCEDURE DEFINES THE PORT B OF 8155 TO INPUT IF STATUS = I AND TO
                  OUTPUT IF STATUS = O.*/

188   2         PORT_B:PROCEDURE(STATUS);
189   2         DECLARE STATUS BYTE;

190   2         IF STATUS='I' THEN COMMAND_REGISTER_COPY=COMMAND_REGISTER_COPY AND 11111101B;
192   2         IF STATUS='O' THEN COMMAND_REGISTER_COPY=COMMAND_REGISTER_COPY OR 00000010B;
194   2         COMMAND_REGISTER=COMMAND_REGISTER_COPY;
195   2         RETURN;
196   1         END PORT_B;

                /*-------------------------------------------------------------------*/
                /*THIS PROCEDURE DEFINES THE PORT C OF 8155 TO ALT1,ALT2,ALT3,ALT4 IF STATUS
                  IS 1,2,3,4 RESPECTIVELY.*/

197   2         PORT_C: PROCEDURE(STATUS);
198   2         DECLARE STATUS BYTE;

199   2         IF STATUS='1' THEN COMMAND_REGISTER_COPY=COMMAND_REGISTER_COPY AND 11110011B;
201   2         IF STATUS='2' THEN COMMAND_REGISTER_COPY=COMMAND_REGISTER_COPY OR 00001100B;
203   3         IF STATUS='3' THEN DO;
205   3            COMMAND_REGISTER_COPY=COMMAND_REGISTER_COPY OR 00000100B;
206   3            COMMAND_REGISTER_COPY=COMMAND_REGISTER_COPY AND 11110111B;
207   3         END;
208   3         IF STATUS='4' THEN DO;
210   3            COMMAND_REGISTER_COPY=COMMAND_REGISTER_COPY OR 00001000B;
211   3            COMMAND_REGISTER_COPY=COMMAND_REGISTER_COPY AND 11111011B;
212   3         END;
213   2         COMMAND_REGISTER=COMMAND_REGISTER_COPY;
214   2         RETURN;
215   1         END PORT_C;

                /*-------------------------------------------------------------------*/
                /*THIS PROCEDURE ENABLES OR DISABLES THE PORT A INTERRUPT.*/

216   2         INTERRUPT_PORT_A:PROCEDURE(OPTION);

217   2         DECLARE OPTION BYTE;
218   2         IF OPTION='E' THEN COMMAND_REGISTER_COPY=COMMAND_REGISTER_COPY OR 00010000B;
220   2         IF OPTION='D' THEN COMMAND_REGISTER_COPY=COMMAND_REGISTER_COPY AND 11101111B;
222   2         COMMAND_REGISTER=COMMAND_REGISTER_COPY;
223   2         RETURN;
224   1         END INTERRUPT_PORT_A;

                /*-------------------------------------------------------------------*/
                /*THIS PROCEDURE ENABLES OR DISABLES THE PORT B INTERRUPT.*/

225   2         INTERRUPT_PORT_B:PROCEDURE(OPTION);
226   2         DECLARE OPTION BYTE;

227   2         IF OPTION='E' THEN COMMAND_REGISTER_COPY=COMMAND_REGISTER_COPY OR 00100000B;
229   2         IF OPTION='D' THEN COMMAND_REGISTER_COPY=COMMAND_REGISTER_COPY AND 11011111B;
231   2         COMMAND_REGISTER=COMMAND_REGISTER_COPY;
232   2         RETURN;
233   1         END INTERRUPT_PORT_B;
```

```
                /*-------------------------------------------------------------------*/
                /*THIS PROCEDURE SETS THE TIMER COMMAND TO MODE
                  0=NOP
                  1=STOP
                  2=STOP AFTER TIMER COUNT REACHED
                  3=START.*/

234   2         TIMER_COMMAND:PROCEDURE(OPTION);
235   2         DECLARE OPTION BYTE;

236   2         IF OPTION='0' THEN COMMAND_REGISTER_COPY=COMMAND_REGISTER_COPY AND 00111111B;
238   2         IF OPTION='3' THEN COMMAND_REGISTER_COPY=COMMAND_REGISTER_COPY OR 11000000B;
240   3         IF OPTION='1' THEN DO;
242   3            COMMAND_REGISTER_COPY=COMMAND_REGISTER_COPY AND 01111111B;
243   3            COMMAND_REGISTER_COPY=COMMAND_REGISTER_COPY OR 01000000B;
244   3         END;
245   3         IF OPTION='2' THEN DO;
247   3            COMMAND_REGISTER_COPY=COMMAND_REGISTER_COPY AND 10111111B;
248   3            COMMAND_REGISTER_COPY=COMMAND_REGISTER_COPY OR 10000000B;
249   3         END;
250   2         COMMAND_REGISTER=COMMAND_REGISTER_COPY;
251   2         RETURN;
252   1         END TIMER_COMMAND;

                /*-------------------------------------------------------------------*/
                /*THIS PROCEDURE LETS THE USER PROGRAM THE COMMAND REGISTER OF THE 8155.*/

253   2         COMMAND_REG:PROCEDURE;
254   2         DECLARE ANSWER1 BYTE;
255   2         DECLARE ANSWER2 BYTE;
256   2         COMMAND_REGISTER_COPY=0H;

257   3         HERE: DO;

259   3         CALL PRINT(.(CRLF,'P=PORT,T=TIMER  ', 0) );
259   3         ANSWER1=GET_CHAR;

260   3         IF ANSWER1='Q' THEN GO TO OUT;
262   4         ELSE DO;
263   5         IF ANSWER1='P' THEN DO;
265   5            CALL PRINT(.(';WHICH PORT  ', 0) );
266   5            ANSWER1=GET_CHAR;

267   6            IF ANSWER1='C' THEN DO;
269   6               CALL PRINT(.(CRLF,'ALT1=1,ALT2=2,ALT3=3,ALT4=4  ', 0) );
270   6               ANSWER2=GET_CHAR;
271   6               CALL PORT_C(ANSWER2);
272   6            END;

273   6            ELSE DO;
274   6               CALL  PRINT(.(CRLF,'I=INPUT,O=OUTPUT,R=INTERRUPT  ', 0) );
275   6                ANSWER2=GET_CHAR;

276   7               IF ANSWER2='R' THEN DO;
278   7                  CALL PRINT(.(CRLF,'E=ENABLE, D=DISABLE  ', 0) );
279   7                  ANSWER2=GET_CHAR;
```

```
280  7            IF ANSWER1='A' THEN CALL INTERRUPT_PORT_A(ANSWER2);
282  7            IF ANSWER1='B' THEN CALL INTERRUPT_PORT_B(ANSWER2);
284  7          END;


285  7          ELSE DO;
286  7            IF ANSWER1='A' THEN CALL PORT_A(ANSWER2);
288  7            IF ANSWER1='B' THEN CALL PORT_B(ANSWER2);
290  7          END;
291  6        END;
292  5      END;


293  5      ELSE DO;
294  6        IF ANSWER1='T' THEN DO;
296  6        CALL PRINT(.(CRLF,'NOP=0,STOP=1,STOP AFTER TC=2,START=3  ', 0) );
297  6        ANSWER2=GET_CHAR;
298  6        CALL TIMER_COMMAND(ANSWER2);
299  6        END;
300  5      ELSE GO TO HERE;
301  5      END;

302  4      END;
303  3      END;
304  2      OUT:RETURN;
305  1      END COMMAND_REG;

         /*-------------------------------------------------------------------*/
         /*THIS PROCEDURE LETS THE USER CHOOSE A COMMAND FOR THE CPU.*/

306  2      CHOOSE_COMMAND:PROCEDURE;
307  2      DECLARE COMMAND WORD;
308  2      DECLARE (HIGHCOM, LOWCOM) BYTE AT (.COMMAND);
309  2      DECLARE COMMAND_TABLE(3) WORD CONSTANT('CR','LD','GO');
310  2      DECLARE ORDER BYTE;

311  2      TAKE_COMMAND:CALL PRINT(.(CRLF,'!', 0) );
312  2      HIGHCOM=GET_CHAR;
313  2      LOWCOM=GET_CHAR;
314  2      ORDER=0;
315  3      DO WHILE COMMAND <> COMMAND_TABLE(ORDER);
316  3      ORDER=ORDER+1;
317  3      IF ORDER=3H THEN GO TO CHOOSE;
319  3      END;
320  3      CHOOSE:DO CASE ORDER;
321  3        CALL COMMAND_REG;
322  3        CALL LOAD;
323  3        CALL PRINT(.('THROUGH IOP', 0));
324  3        CALL PRINT(.('CHECK COMMAND', 0));
325  3      END;
326  2      ECHO=GET_CHAR;
327  2      IF ECHO<>'Q' THEN GOTO TAKE_COMMAND;
329  2      RETURN;
330  1      END CHOOSE_COMMAND;
```

```
                /*************************************************************************/
                /*                          THE MAIN PROGRAM                             */
                /*************************************************************************/

331   2     INIT:DO; /*INITIALIZE THE CPU*/
332   2          TH1=11111110B; /*INSERT -2 TO TH1*/
333   2          SCON=01000010B;/* THE SERIAL CONTROL INITIALIZED TO 8-BIT MODE1*/
334   2          TMOD=00100000B;/*TIMER1 INITIALIZED TO 8_BIT AUTORELOAD MODE*/
335   2          P1=0;/*INITIALIZE THE SERIAL PORT*/
336   2          ES=0;/*DISABLE THE SERIAL INTERRUPT*/
337   2          EA=1;/*ENABLE THE INTERRUPTS*/
338   2          TR1=1;
339   2          EX1=1;
340   2          MODE_FLAG=0;
341   1     END INIT;


342   2     POLL: DO;
343   2               CPU_COMMUN=0;
344   3     WAIT: DO WHILE NOT CPU_COMMUN;
345   3               END;
346   2               CPU_COMMUN=0;
347   2               CPU_IOP=0;
348   2               IF GOLOCATION<>0 THEN CALL USER_GO;
350   2               REN=1;
351   2               EX1=0;
352   2               MODE=0;
353   2               CALL PRINT(.(125, 1CH, 0) );
354   2               CALL PRINT(.(CRLF,0) );
355   2               CALL PRINT(.('ENTER COMMAND FOR CPU: ', 0) );
356   2               CALL CHOOSE_COMMAND;

357   2               CALL PRINT(.(CRLF,0) );
358   3               DO WHILE NOT TI;
359   3               END;
360   2               INTERRUPT_ACKN=0;        /*RETURN THE SERIAL LINES TO THE IOP*/
361   2               ACC=0;
362   2               ACC=0;
363   2               INTERRUPT_ACKN=1;
364   2               GO TO INIT;
365   1     END POLL;

                /*************************************************************************/
                /*                          END OF MAIN PROGRAM                          */
                /*************************************************************************/


366   1     END MONITOR;
```

```
MODULE INFORMATION:             (STATIC+OVERLAYABLE)
    CODE SIZE                   = 0480H      1152D
    CONSTANT SIZE               = 0132H       306D
    DIRECT VARIABLE SIZE        =   0EH+0AH    14D+ 10D
    INDIRECT VARIABLE SIZE      =   00H+00H     0D+  0D
    BIT SIZE                    =   03H+00H     3D+  0D
    BIT-ADDRESSABLE SIZE        =   00H+00H     0D+  0D
    AUXILIARY VARIABLE SIZE     = 0000H         0D
    MAXIMUM STACK SIZE          = 0011H        17D
    REGISTER-BANK(S) USED:        0

    540 LINES READ
    0 PROGRAM ERROR(S)

END OF PL/M-51 COMPILATION
```

REFERENCES

1. Intel Corporation, Microcontroller Handbook, 1984

2. Intel Corporation, Microprocessor and Peripheral Handbook, 1983

3. Intel Corporation, PL/M-51 User's Guide, 1983